

ASSIGNMENT – Electronic Gadgets
Swathi Baskaran (Superset ID: 5282910)

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:
 - o 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.

- o 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:

- o 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.

- o 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:

- o 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:

- o 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.

Query:

```
from exception.exceptionHandling import DatabaseError

class Customer():

    def __init__(self, CustomerID, FirstName, LastName, Email, Phone,
Address):
        self.__CustomerID = CustomerID
        self.__FirstName = FirstName
        self.__LastName = LastName
        self.__Email = Email
        self.__Phone = Phone
        self.__Address = Address

    #Setting getters for every variable involved

    @property
    def CustomerID(self):
        return self.__CustomerID

    @property
    def FirstName(self):
        return self.__FirstName

    @property
    def LastName(self):
        return self.__LastName

    @property
```

```
def Email(self):
    return self.__Email

@property
def Phone(self):
    return self.__Phone

@property
def Address(self):
    return self.__Address

#Setting variables along with validation
@FirstName.setter
def FirstName(self, value):
    if not value:
        raise ValueError ("[First Name] field cannot be empty..")
    if not isinstance(value, str):
        raise ValueError ("First Name should be a string..")
    self.__FirstName = value

@LastName.setter
def LastName(self, value):
    if not value:
        raise ValueError("[Last Name] field cannot be empty..")
    if not isinstance(value, str):
        raise ValueError("Last Name should be a string")
```

```
    self.__LastName = value
```

```
@Email.setter
```

```
def Email(self, value):
```

```
    if "@" not in value or "." not in value:
```

```
        raise ValueError ("Invalid Email address")
```

```
    self.__Email = value
```

```
@Phone.setter
```

```
def Phone(self, value):
```

```
    if value is None:
```

```
        raise ValueError("Contact Number field cannot be empty")
```

```
    if not str(value).isdigit() or len(str(value))!= 10:
```

```
        raise ValueError("Contact Number must contain 10 digits")
```

```
    self.__Phone = str(value)
```

```
@Address.setter
```

```
def Address(self, value):
```

```
    if not value:
```

```
        raise ValueError("Address field cannot be empty")
```

```
    self.__Address = value
```

```
def CalculateTotalOrders(self):
```

```
    try:
```

```
        from dao.order_dao import OrderDAO
```

```
        countOfOrders = OrderDAO.get_order_count(self.__CustomerID)
```

```
        totalAmountSpent =
```

```
        OrderDAO.get_customer_total_spent(self.__CustomerID)
```

```
    return countOfOrders, totalAmountSpent
except Exception as e:
    raise DatabaseError (f"Database Interaction Failed: {e.msg}") from e
```

```
def GetCustomerDetails(self):
    return f"          Customer ID: {self.__CustomerID}
\n          First Name: {self.__FirstName} \n          LastName:
{self.__LastName} \n          Email: {self.__Email}
\n          Contact Number: {self.__Phone} \n          Residential
Address: {self.__Address}"
```

```
def UpdateCustomerInfo(self, email = None, Phone = None, Address =
None):
```

```
    if email:
        self.Email = email
    if Phone:
        self.Phone = Phone
    if Address:
        self.Address = Address
```

Output:

customer.py:

```

from exception.exceptionHandling import DatabaseError

class Customer():
    def __init__(self, CustomerID, FirstName, LastName, Email, Phone, Address):
        self._CustomerID = CustomerID
        self._FirstName = FirstName
        self._LastName = LastName
        self._Email = Email
        self._Phone = Phone
        self._Address = Address

    #Setting getters for every variable involved
    @property
    def CustomerID(self):
        return self._CustomerID

    @property
    def FirstName(self):
        return self._FirstName

    @property
    def LastName(self):
        return self._LastName

    @property
    def Email(self):
        return self._Email

    @property
    def Phone(self):
        return self._Phone

    @property
    def Address(self):
        return self._Address

```

TechShopDB > entity > customer.py > Customer

```

3   class Customer():
3>
38     #Setting variables along with validation
39     @FirstName.setter
40     def FirstName(self, value):
41         if not value:
42             raise ValueError ("[First Name] field cannot be empty..")
43         if not isinstance(value, str):
44             raise ValueError ("First Name should be a string..")
45         self._FirstName = value
46
47     @LastName.setter
48     def LastName(self, value):
49         if not value:
50             raise ValueError("[Last Name] field cannot be empty..")
51         if not isinstance(value, str):
52             raise ValueError("Last Name should be a string")
53         self._LastName = value
54
55     @Email.setter
56     def Email(self, value):
57         if "@" not in value or "." not in value:
58             raise ValueError ("Invalid Email address")
59         self._Email = value
60
61     @Phone.setter
62     def Phone(self, value):
63         if value is None:
64             raise ValueError("Contact Number field cannot be empty")
65         if not str(value).isdigit() or len(str(value))!= 10:
66             raise ValueError("Contact Number must contain 10 digits")
67         self._Phone = str(value)
68
69     @Address.setter
70     def Address(self, value):
71         if not value:
72             raise ValueError("Address field cannot be empty")
73         ...

```

Ln 54, Col 9 | Spaces: 4 | UTF-8 | CRLF | Python | 3.13.1 64-bit | Go Live |

```
TechShopDB > entity > customer.py > Customer
  3     class Customer():
  4         def __init__(self, CustomerID, FirstName, LastName, Email, Phone, Address):
  5             self._CustomerID = CustomerID
  6             self._FirstName = FirstName
  7             self._LastName = LastName
  8             self._Email = Email
  9             self._Phone = Phone
 10            self._Address = Address
 11
 12        @CustomerID.setter
 13        def CustomerID(self, value):
 14            self._CustomerID = value
 15
 16        @FirstName.setter
 17        def FirstName(self, value):
 18            if not value:
 19                raise ValueError("First Name field cannot be empty")
 20            self._FirstName = value
 21
 22        @LastName.setter
 23        def LastName(self, value):
 24            if not value:
 25                raise ValueError("Last Name field cannot be empty")
 26            self._LastName = value
 27
 28        @Email.setter
 29        def Email(self, value):
 30            if not value:
 31                raise ValueError("Email field cannot be empty")
 32            self._Email = value
 33
 34        @Phone.setter
 35        def Phone(self, value):
 36            self._Phone = str(value)
 37
 38        @Address.setter
 39        def Address(self, value):
 40            if not value:
 41                raise ValueError("Address field cannot be empty")
 42            self._Address = value
 43
 44    def CalculateTotalOrders(self):
 45        try:
 46            from dao.order_dao import OrderDAO
 47            countOfOrders = OrderDAO.get_order_count(self._CustomerID)
 48            totalAmountSpent = OrderDAO.get_customer_total_spent(self._CustomerID)
 49            return countOfOrders, totalAmountSpent
 50        except Exception as e:
 51            raise DatabaseError(f"Database Interaction Failed: {e.msg}") from e
 52
 53    def GetCustomerDetails(self):
 54        return f"Customer ID: {self._CustomerID} \nFirst Name: {self._FirstName} \nLast Name: {self._LastName} \nEmail: {self._Email} \nPhone: {self._Phone} \nAddress: {self._Address}"
 55
 56    def UpdateCustomerInfo(self, email = None, Phone = None, Address = None):
 57        if email:
 58            self.Email = email
 59        if Phone:
 60            self.Phone = Phone
 61        if Address:
 62            self.Address = Address
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
```

product.py:

```
TechShopDB > entity > product.py > Product
  1     class Product():
  2         def __init__(self, ProductID, ProductName, Description, Price):
  3             self._ProductID = ProductID
  4             self._ProductName = ProductName
  5             self._Description = Description
  6             self._Price = Price
  7
  8         #Getting the values for these attributes
  9         @property
 10         def ProductID(self):
 11             return self._ProductID
 12
 13         @property
 14         def ProductName(self):
 15             return self._ProductName
 16
 17         @property
 18         def Description(self):
 19             return self._Description
 20
 21         @property
 22         def Price(self):
 23             return self._Price
 24
 25         #Setting the variables and checking their validity
 26         @ProductName.setter
 27         def ProductName(self, value):
 28             if not value:
 29                 raise ValueError("[Product Name] field cannot be empty")
 30             self._ProductName = value
 31
 32         @Description.setter
 33         def Description(self, value):
 34             self._Description = value
 35
 36         @Price.setter
 37         def Price(self, value):
```

```

55     @Price.setter
56     def Price(self, value):
57         if value < 0:
58             raise ValueError("The price of a product cannot be negative")
59         self.__Price = value
60
61     def GetProductDetails(self):
62         return f"Product ID: {self.__ProductID} \nProduct Name: {self.__ProductName} \n Description: {self.__Description} \nPrice: Rs.{self.__P
63
64     def UpdateProductInfo(self, ProductName = None, Price = None, Description = None):
65         if ProductName:
66             self.ProductName = ProductName
67         if Description:
68             self.Description = Description
69         if Price is not None:
70             self.Price = Price
71
72
73
74

```

Ln 17, Col 14 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit ⚡ Go Live ⌂

orders.py:

```

TechShopDB > entity > orders.py > Orders > TotalAmount
1     from datetime import date
2
3     class Orders():
4         def __init__(self, OrderID, Customer, OrderDate = None, TotalAmount = None):
5             self.__OrderID = OrderID
6             self.__Customer = Customer
7             self.__OrderDate = OrderDate if OrderDate else date.today()
8             self.__TotalAmount = TotalAmount
9             self.__Status = "Pending"
10
11     #Getters for the attributes
12     @property
13     def OrderID(self):
14         return self.__OrderID
15
16     @property
17     def Customer(self):
18         return self.__Customer
19
20     @property
21     def OrderDate(self):
22         return self.__OrderDate
23
24     @property
25     def TotalAmount(self):
26         return self.__TotalAmount
27
28     @property
29     def Status(self):
30         return self.__Status
31
32     #Setters for the attributes with validity
33
34     @Customer.setter
35     def TotalAmount(self, value):
36         if value < 0:
37             raise ValueError ("Total Amount cannot be negative..")

```

Ln 37, Col 65 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit ⚡ Go Live ⌂

```

33     @Customer.setter
34     def TotalAmount(self, value):
35         if value < 0:
36             raise ValueError ("Total Amount cannot be negative..")
37         self.__TotalAmount = value
38
39
40     def CalculateTotalAmount(self):
41         OrderDetails = OrderDetailsDAO.get_order_details_by_order(self.__OrderID)
42         return sum(
43             detail.Quantity * (detail.Product.Price * (1 - detail.Discount))
44             for detail in OrderDetails
45         )
46
47     def UpdateOrderStatus(self, newStatus):
48         valid_statuses = ["Pending", "Processing", "Shipped", "Delivered", "Cancelled"]
49         if newStatus not in valid_statuses:
50             raise ValueError (f"The status of the product should be one of these : {', '.join (valid_statuses)}")
51         self.__Status = newStatus

```

Ln 51, Col 65 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit ⚡ Go Live ⌂

orderDetails.py:

```
TechShopDB > entity > orderDetails.py > OrderDetails > Order
 1  class OrderDetails():
 2      def __init__(self, OrderDetailID, Order, Product, Quantity):
 3          self.__OrderDetailID = OrderDetailID
 4          self.__Order = Order
 5          self.__Product = Product
 6          self.__Quantity = Quantity
 7          self.__Discount = 0.0
 8
 9      #Getters for the attributes
10      @property
11      def OrderDetailID(self):
12          return self.__OrderDetailID
13
14      @property
15      def Order(self):
16          return self.__Order
17
18      @property
19      def Product(self):
20          return self.__Product
21
22      @property
23      def Quantity(self):
24          return self.__Quantity
25
26      @property
27      def Discount(self):
28          return self.__Discount
29
30      #Setters for the Attributes
31      @Quantity.setter
32      def Quantity(self, value):
33          if value <= 0:
34              raise ValueError ("Quantity must be greater than 0")
35          self.__Quantity = value
36
37      # @Discount.setter
38
39      # def Discount(self, value):
40      #     if value < 0:
41      #         raise ValueError("Discount cannot be lesser than 0")
42      #     self.__Discount = value
43
44      def CalculateSubtotal(self):
45          return self.__Product.Price * self.__Quantity * (1 - self.__Discount)
46
47      def GetOrderDetailInfo(self):
48          return f"OrderDetail ID: {self.__OrderDetailID} \nProduct Name: {self.__Product.ProductName} \nProduct Price: {self.__Product.Price:.2f}"
49
50      def UpdateQuantity(self, newQuantity):
51          self.__Quantity = newQuantity
52
53      def AddDiscount(self, addedDiscount):
54          if not 0 <= addedDiscount <= 100:
55              raise ValueError ("Discount Percentage must be in the range of 0 - 100...")
56          self.__Discount += (addedDiscount/100)
57
```

Ln 15, Col 21 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
30
31      # @Discount.setter
32      # def Discount(self, value):
33      #     if value < 0:
34      #         raise ValueError("Discount cannot be lesser than 0")
35      #     self.__Discount = value
36
37      def CalculateSubtotal(self):
38          return self.__Product.Price * self.__Quantity * (1 - self.__Discount)
39
40      def GetOrderDetailInfo(self):
41          return f"OrderDetail ID: {self.__OrderDetailID} \nProduct Name: {self.__Product.ProductName} \nProduct Price: {self.__Product.Price:.2f}"
42
43      def UpdateQuantity(self, newQuantity):
44          self.__Quantity = newQuantity
45
46      def AddDiscount(self, addedDiscount):
47          if not 0 <= addedDiscount <= 100:
48              raise ValueError ("Discount Percentage must be in the range of 0 - 100...")
49          self.__Discount += (addedDiscount/100)
50
51      def GetTotalPrice(self):
52          return self.__Product.Price * self.__Quantity * (1 - self.__Discount)
53
54      def GetSubtotalPrice(self):
55          return self.__Product.Price * self.__Quantity
56
57
```

inventory.py:

TechShopDB > entity > inventory.py > Inventory > ListLowStockProducts

```
3 class Inventory():
4
5     @LastStockUpdate.setter
6     def LastStockUpdate(self, value):
7         if not isinstance(value, date):
8             raise ValueError("Last Stock Update must be in date format..")
9         self.__LastStockUpdate = value
10
11    def GetProduct(self):
12        return self.__Product
13
14    def GetQuantityInStock(self):
15        return self.__QuantityInStock
16
17    def AddToInventory(self, quantity):
18        if quantity <= 0:
19            raise ValueError ("Quantity to be added must be larger than 0..")
20        self.__QuantityInStock += quantity
21        self.__LastStockUpdate = "CURRENT_DATE"
22
23    def RemoveFromInventory(self, quantity):
24        if self.__QuantityInStock < quantity:
25            raise ValueError ("Insufficient stock in the inventory")
26        if quantity <= 0:
27            raise ValueError ("Enter a positive quantity to be removed from the inventory")
28        self.__QuantityInStock -= quantity
29        self.__LastStockUpdate = "CURRENT_TIMESTAMP"
30
31    def UpdateStockQuantity(self, newQuantity):
32        if newQuantity < 0:
33            raise ValueError ("Enter a valid quantity..")
34        self.__QuantityInStock = newQuantity
35        self.__LastStockUpdate = "CURRENT_TIMESTAMP"
36
37    def IsProductAvailable(self, quantityToCheck):
38        if quantityToCheck < 0:
39            raise ValueError ("Enter a valid quantity..")
```

```
71
72     def GetInventoryValue(self):
73         return self._Product.Price * self._QuantityInStock
74
75     def ListLowStockProducts(self, threshold):
76         from dao.inventory_dao import InventoryDAO
77         return InventoryDAO.get_products_by_quantity([threshold])
78
79     def ListOutOfStockProducts(self, threshold = 1):
80         from dao.inventory_dao import InventoryDAO
81         return InventoryDAO.get_products_by_quantity(threshold)
82
83     def ListAllProducts(self):
84         return f"Product Name: {self._Product.ProductName} \nQuantity: {self._QuantityInStock}"
```

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.

- o Exception Handling: Handle payment-specific exceptions (e.g., PaymentFailedException)

and initiate retry or cancellation processes.

- File I/O (e.g., Logging):

- o Challenge: Logging errors and events to files or databases.

- o Scenario: When an error occurs during data persistence (e.g., writing a log entry).

- o Exception Handling: Handle file I/O exceptions (e.g., IOException) and log them

appropriately.

- Database Access:

- o Challenge: Managing database connections and queries.

- o Scenario: When executing a SQL query and the database is offline.

- o Exception Handling: Handle database-specific exceptions (e.g., SQLException) and

implement connection retries or failover mechanisms.

- Concurrency Control:

- o Challenge: Preventing data corruption in multi-user scenarios.

- o Scenario: When two users simultaneously attempt to update the same order.

- o Exception Handling: Implement optimistic concurrency control and handle ConcurrencyException by notifying users to retry.

- Security and Authentication:

- o Challenge: Ensuring secure access and handling unauthorized access attempts.

- o Scenario: When a user tries to access sensitive information without proper authentication.

- o Exception Handling: Implement custom AuthenticationException and AuthorizationException to handle security-related issues.

Query:

```
class DatabaseError(Exception):  
    def __init__(self, message = "Database Connection Failed"):  
        super().__init__(message)  
        #self.errorcode = 500
```

```
class CustomerNotFoundError(DatabaseError):  
    def __init__(self, CustomerID):  
        super().__init__(f"Customer {CustomerID} not found")  
        #self.errorcode = 404
```

```
class ProductNotFoundError(DatabaseError):  
    def __init__(self, ProductID):  
        super().__init__(f"Product {ProductID} not found")  
        #self.errorcode = 404
```

```
class OrderNotFoundError(DatabaseError):  
    def __init__(self, OrderID):  
        super().__init__(f"Order {OrderID} not found")  
        #self.errorcode = 404
```

```
class OrdersNotPlacedError(DatabaseError):  
    def __init__(self, CustomerID):  
        super().__init__(f"Customer {CustomerID} hasn't placed any  
orders")  
        #self.errorcode = 404
```

```
class InsufficientStockError(Exception):
```

```

        def __init__(self, ProductID, ReqQuantity, AvailableQuantity):
            super().__init__(f"Only {AvailableQuantity} stock units are
available for Product {ProductID} \n(Requested Stock Quantity:
{ReqQuantity})")

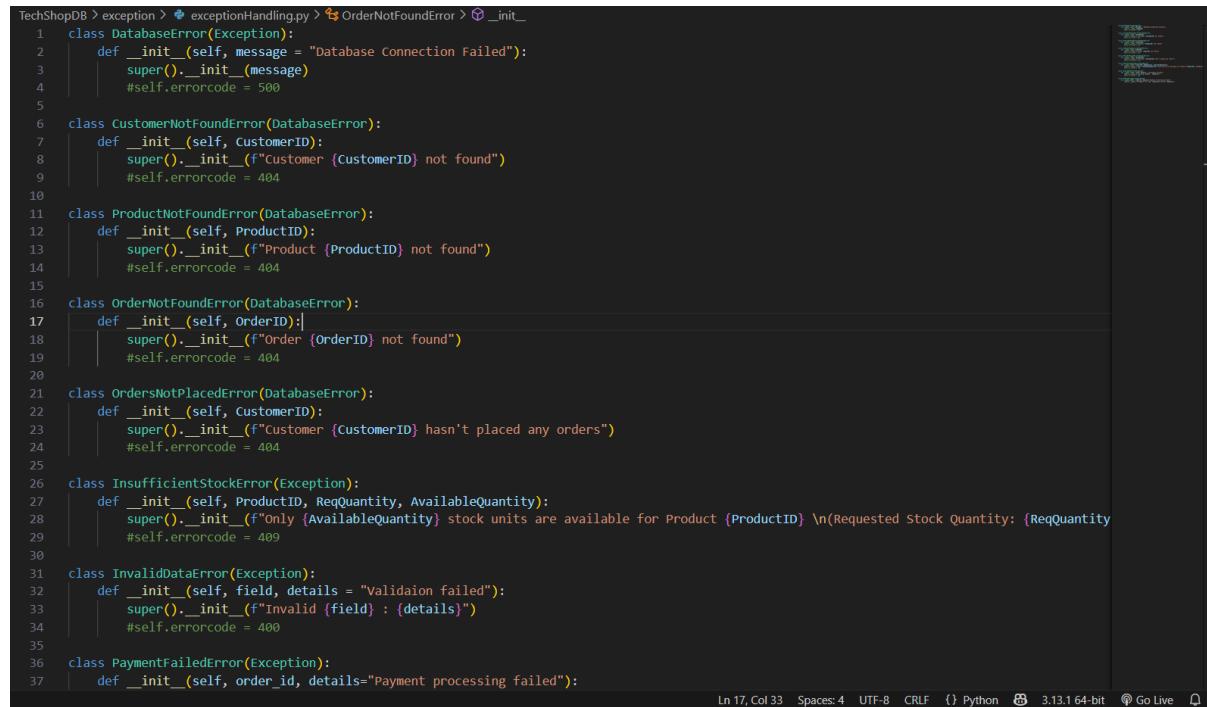
            #self.errorcode = 409

class InvalidDataError(Exception):
    def __init__(self, field, details = "Validaion failed"):
        super().__init__(f"Invalid {field} : {details}")
        #self.errorcode = 400

class PaymentFailedError(Exception):
    def __init__(self, order_id, details="Payment processing failed"):
        super().__init__(f"Payment for order {order_id} failed: {details}")

```

Output:



```

TechShopDB > exception > exceptionHandling.py > OrderNotFoundError > __init__
  1  class DatabaseError(Exception):
  2      def __init__(self, message = "Database Connection Failed"):
  3          super().__init__(message)
  4          #self.errorcode = 500
  5
  6  class CustomerNotFoundError(DatabaseError):
  7      def __init__(self, CustomerID):
  8          super().__init__(f"Customer {CustomerID} not found")
  9          #self.errorcode = 404
 10
 11 class ProductNotFoundError(DatabaseError):
 12     def __init__(self, ProductID):
 13         super().__init__(f"Product {ProductID} not found")
 14         #self.errorcode = 404
 15
 16 class OrderNotFoundError(DatabaseError):
 17     def __init__(self, OrderID):
 18         super().__init__(f"Order {OrderID} not found")
 19         #self.errorcode = 404
 20
 21 class OrdersNotPlacedError(DatabaseError):
 22     def __init__(self, CustomerID):
 23         super().__init__(f"Customer {CustomerID} hasn't placed any orders")
 24         #self.errorcode = 404
 25
 26 class InsufficientStockError(Exception):
 27     def __init__(self, ProductID, ReqQuantity, AvailableQuantity):
 28         super().__init__(f"Only {AvailableQuantity} stock units are available for Product {ProductID} \n(Requested Stock Quantity: {ReqQuantity})")
 29         #self.errorcode = 409
 30
 31 class InvalidDataError(Exception):
 32     def __init__(self, field, details = "Validaion failed"):
 33         super().__init__(f"Invalid {field} : {details}")
 34         #self.errorcode = 400
 35
 36 class PaymentFailedError(Exception):
 37     def __init__(self, order_id, details="Payment processing failed"):

```

Ln 17, Col 33 Spaces: 4 UTF-8 CRLF {} Python Go Live

```
31     class InvalidDataError(Exception):
32         def __init__(self, field, details = "Validation failed"):
33             super().__init__(f"Invalid {field} : {details}")
34             self.errorcode = 400
35
36     class PaymentFailedError(Exception):
37         def __init__(self, order_id, details="Payment processing failed"):
38             super().__init__(f"Payment for order {order_id} failed: {details}")
39
```

Task 6: Collections

- Managing Products List:

- o Challenge: Maintaining a list of products available for sale (List<Products>).

- o Scenario: Adding, updating, and removing products from the list.

- o 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:

- o Challenge: Maintaining a list of customer orders (List<Orders>).

- o Scenario: Adding new orders, updating order statuses, and removing canceled orders.

- o 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:

- o Challenge: Sorting orders by order date in ascending or descending order.

- o Scenario: Retrieving and displaying orders based on specific date ranges.

- o 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:

- o Challenge: Managing product inventory with a SortedList based on product IDs.

- o Scenario: Tracking the quantity in stock for each product and quickly retrieving inventory information.
- o 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:
- o Challenge: Ensuring that inventory is updated correctly when processing orders.
- o Scenario: Decrementing product quantities in stock when orders are placed.
- o Solution: Implement a method to update inventory quantities when orders are processed. Handle exceptions for insufficient stock.
 - Product Search and Retrieval:
- o Challenge: Implementing a search functionality to find products based on various criteria (e.g., name, category).
- o Scenario: Allowing customers to search for products.
- o Solution: Implement custom search methods using LINQ queries on the List<Products> collection. Handle exceptions for invalid search criteria.
 - Duplicate Product Handling:
- o Challenge: Preventing duplicate products from being added to the list.
- o Scenario: When a product with the same name or SKU is added.
- o 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:
- o Challenge: Managing a list of payment records for orders (List<PaymentClass>).
- o Scenario: Recording and updating payment information for each order.

- o 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:
- o Challenge: Managing the relationship between OrderDetails and Products.
- o Scenario: Ensuring that order details accurately reflect the products available in the inventory.
- o Solution: Implement methods to validate product availability in the inventory before adding order details. Handle exceptions for unavailable products.

Query:

customer_dao.py:

```
from util.DB_Connections import DBConnections
from entity.customer import Customer
from exception.exceptionHandling import DatabaseError,
CustomerNotFoundError, InvalidDataError
import mysql.connector
```

```
class CustomerDAO():
```

```
    @staticmethod
    def get_customer(CustomerID):
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary = True)
```

```
query = "SELECT * FROM Customers WHERE CustomerID = %s"
cursor.execute(query,(CustomerID,))
result = cursor.fetchone()

if not result:
    raise CustomerNotFoundError (CustomerID)
return Customer (
    CustomerID = result['CustomerID'],
    FirstName = result['FirstName'],
    LastName = result['LastName'],
    Email = result['Email'],
    Phone = result['Phone'],
    Address = result['Address']
)

except mysql.connector.Error as e:
    raise DatabaseError (f'Database Interaction Failed, {e.msg}') from e
```

```
finally:
    if 'cursor' in locals():
        cursor.close()

@staticmethod
def add_customer(Customer):
    try:
        conn = DBConnections.get_connection()
```

```
cursor = conn.cursor(dictionary = True)

query = """
    INSERT INTO Customers(FirstName, LastName, Email, Phone,
Address)
    VALUES (%s, %s, %s, %s, %s)
"""

cursor.execute(query, (Customer.FirstName, Customer.LastName,
Customer.Email, Customer.Phone, Customer.Address))
conn.commit()

return cursor.lastrowid

except mysql.connector.Error as e:
    conn.rollback()
    if "Duplicate entry" in str(e):
        raise InvalidDataError ("Email", details = "Email already
registered..")
    raise DatabaseError (f"Database Interaction Failed : {e.msg}") from e

finally:
    if 'cursor' in locals():
        cursor.close()

@staticmethod
def update_customer(Customer):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)
```

```
query = """
UPDATE Customers SET Email = %s, Phone = %s, Address = %s
WHERE CustomerID = %s"""

cursor.execute(query, (Customer.Email, Customer.Phone,
Customer.Address, Customer.CustomerID))
conn.commit()
return cursor.lastrowid

except mysql.connector.Error as e:
    conn.rollback()
    raise DatabaseError (f"Database Interaction failed: {e.msg}") from e

finally:
    if 'cursor' in locals():
        cursor.close()

@staticmethod
def get_all_customers():
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        query = "SELECT * FROM Customers ORDER BY CustomerID"

        cursor.execute(query)
        results = cursor.fetchall()
```

```
customers = []
for result in results:
    customers.append(Customer(
        CustomerID = result['CustomerID'],
        FirstName = result['FirstName'],
        LastName = result['LastName'],
        Email = result['Email'],
        Phone = result['Phone'],
        Address = result['Address']
    ))
return customers

except mysql.connector.Error as e:
    raise DatabaseError (f"Database Interaction Failed: {e.msg}") from e
finally:
    if 'cursor' in locals():
        cursor.close()
```

product_dao.py:

```
from util.DB_Connections import DBConnections
from entity.product import Product
from exception.exceptionHandling import DatabaseError,
ProductNotFoundError, InvalidDataError
import mysql.connector
```

```
class ProductDAO():
```

```
@staticmethod  
def get_product(ProductID):  
    try:  
        conn = DBConnections.get_connection()  
        cursor = conn.cursor(dictionary = True)  
  
        query = "SELECT * FROM Products WHERE ProductID = %s"  
        cursor.execute(query,(ProductID,))  
        result = cursor.fetchone()  
  
        if not result:  
            raise ProductNotFoundError(ProductID)  
  
        return Product (  
            ProductID = result['ProductID'],  
            ProductName = result['ProductName'],  
            Description = result['Description'],  
            Price = result['Price']  
        )  
  
    except mysql.connector.Error as e:  
        raise DatabaseError (f'Database Interaction Failed: {e.msg}') from e  
  
    finally:  
        if 'cursor' in locals():  
            cursor.close()
```

```
@staticmethod  
def add_product(Product):  
    try:  
  
        if not Product.ProductName or not Product.Description:  
            raise InvalidDataError("Product", details = "Product name and  
Description are required")  
        if Product.Price <= 0:  
            raise InvalidDataError("Product", details = "Product Price must be  
positive")  
  
        conn = DBConnections.get_connection()  
        cursor = conn.cursor(dictionary = True)  
  
        conn.start_transaction()  
  
        check_query = "SELECT ProductID FROM Products WHERE  
ProductName = %s"  
        cursor.execute(check_query,(Product.ProductName,))  
        if cursor.fetchone():  
            raise InvalidDataError ("Product", details = "Product already exists in  
the database")  
  
        product_query = """  
        INSERT INTO Products(ProductName, Description, Price)  
        VALUES (%s, %s, %s)  
        """  
        cursor.execute(product_query, (Product.ProductName,  
Product.Description, Product.Price))
```

```
product_id = cursor.lastrowid

    inventory_query = """INSERT INTO Inventory(ProductID,
QuantityInStock, LastStockUpdate)
VALUES (%s, 0, CURRENT_DATE)"""
    cursor.execute(inventory_query, (product_id,))

    conn.commit()
    return product_id

except mysql.connector.Error as e:
    if conn:
        conn.rollback()
    if "Duplicate entry" in str(e):
        raise InvalidDataError ("Product", details = "Product already exists")
    raise DatabaseError (f'Database Interaction Failed : {e.msg}') from e

finally:
    if conn and conn.is_connected():
        if 'cursor' in locals():
            cursor.close()

@staticmethod
def update_product(product_obj):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)
```

```
query = """
UPDATE Products SET ProductName = %s, Description = %s, Price =
%s
WHERE ProductID = %s"""

cursor.execute(query, (product_obj.ProductName,
product_obj.Description, product_obj.Price, product_obj.ProductID))
conn.commit()
return cursor.rowcount > 0

except mysql.connector.Error as e:
    conn.rollback()
    raise DatabaseError (f'Database Interaction failed: {e.msg}') from e

finally:
    if 'cursor' in locals():
        cursor.close()

@staticmethod
def delete_product(productId):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        inventory_query = "DELETE FROM Inventory WHERE ProductID =
%s"
        cursor.execute(inventory_query,(productId,))


```

```
product_query = "DELETE FROM Products WHERE ProductID = %s"
cursor.execute(product_query,(productId,))
conn.commit()

return cursor.rowcount > 0

except mysql.connector.Error as e:
    raise DatabaseError (f"Database Interaction Failed: {e.msg}") from e
except Exception as e:
    print(f"Unexpected error: {e}")

finally:
    if 'cursor' in locals():
        cursor.close()

@staticmethod
def get_all_products():

    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        query = "SELECT * FROM Products ORDER BY ProductID"
        cursor.execute(query)
        results = cursor.fetchall()

        products = []
        for result in results:
```

```
        products.append(Product(
            ProductID = result['ProductID'],
            ProductName = result['ProductName'],
            Description = result['Description'],
            Price = result['Price']
        )))
    return products

except mysql.connector.Error as e:
    raise DatabaseError(f"Database Interaction Failed: {e.msg}") from e
```

```
finally:
```

```
    if 'cursor' in locals():
        cursor.close()
```

order_dao.py:

```
from util.DB_Connections import DBConnections
from entity.orders import Orders
from exception.exceptionHandling import OrderNotFoundError,
OrdersNotPlacedError, DatabaseError
import mysql.connector
from datetime import date
```

```
class OrderDAO():
```

```
    @staticmethod
    def get_order(OrderID):
```

```
try:
    conn = DBConnections.get_connection()
    cursor = conn.cursor(dictionary = True)

    query = "SELECT * FROM Orders WHERE OrderID = %s"
    cursor.execute(query,(OrderID,))
    result = cursor.fetchone()

    if not result:
        raise OrderNotFoundError(OrderID)

    return Orders (
        OrderID = result['OrderID'],
        Customer = result['CustomerID'],
        OrderDate = result['OrderDate'],
        TotalAmount = result['TotalAmount'],
        Status = result['Status']
    )

except mysql.connector.Error as e:
    raise DatabaseError (f'Database Interaction Failed: {e}') from e

finally:
    if 'cursor' in locals():
        cursor.close()
```

```
@staticmethod  
  
def create_order(CustomerID, TotalAmount):  
    try:  
        conn = DBConnections.get_connection()  
        cursor = conn.cursor(dictionary = True)  
  
        query = """  
            INSERT INTO Orders(CustomerID, OrderDate, TotalAmount, Status)  
            VALUES(%s, %s, %s, 'Pending')  
        """  
  
        cursor.execute(query, (CustomerID, date.today(), TotalAmount))  
        conn.commit()  
  
        return cursor.lastrowid  
  
  
    except mysql.connector.Error as e:  
        conn.rollback()  
        raise DatabaseError (f'Database Interaction Failed: {e}') from e  
  
  
    finally:  
        if 'cursor' in locals():  
            cursor.close()  
  
  
@staticmethod  
  
def get_order_count(CustomerID):  
    try:  
        conn = DBConnections.get_connection()  
        cursor = conn.cursor(dictionary = True)
```



```
TotalAmountSpent = cursor.fetchone()

if not TotalAmountSpent:
    return OrdersNotPlacedError(CustomerID)

return TotalAmountSpent

except mysql.connector.Error as e:
    raise DatabaseError(f"Database Interaction Failed: {e.msg}") from e

finally:
    if 'cursor' in locals():
        cursor.close()

@staticmethod
def get_all_orders():

    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        query = """
SELECT Ord.*, Cus.FirstName, Cus.LastName
FROM Orders Ord
INNER JOIN Customers Cus
ON Ord.CustomerID = Cus.CustomerID
ORDER BY Ord.OrderDate DESC, Ord.OrderID ASC
"""

        """
```

```
        cursor.execute(query)
        result = cursor.fetchall()

        if not result:
            raise OrderNotFoundError(f"No orders found: {e}")

        return result

    except mysql.connector.Error as e:
        raise DatabaseError (f'Database Interaction failed: {e.msg}') from e
```

```
finally:
    if 'cursor' in locals():
        cursor.close()
```

orderDetails_dao.py:

```
from util.DB_Connections import DBConnections
from entity.orderDetails import OrderDetails
from entity.product import Product
from entity.orders import Orders
from exception.exceptionHandling import InsufficientStockError,
DatabaseError
import mysql.connector
```

```
class OrderDetailsDAO():
```

```
    @staticmethod
    def add_order_detail(OrderID, ProductID, Quantity):
        try:
```

```
conn = DBConnections.get_connection()
cursor = conn.cursor(dictionary = True)

#Fetching the available stock in the inventory
query = "SELECT QuantityInStock FROM Inventory WHERE
ProductID = %s"
cursor.execute(query, (ProductID,))

stock = cursor.fetchone()

#Checking if there is sufficient amount of stock in the inventory
if stock < Quantity:
    raise InsufficientStockError(ProductID, Quantity, stock)

query = """
INSERT INTO OrderDetails(OrderID, ProductID, Quantity)
VALUES(%s, %s, %s)
"""

cursor.execute(query, (OrderID, ProductID, Quantity))
conn.commit()

#Updating Inventory
query = "UPDATE Inventory SET QuantityInStock = QuantityInStock -
%s WHERE ProductID = %s"
cursor.execute(query, (Quantity, ProductID))
conn.commit()
```

```
except mysql.connector.Error as e:  
    conn.rollback()  
    raise DatabaseError (f"Database Error : {e}") from e  
  
finally:  
    if 'cursor' in locals():  
        cursor.close()  
  
@staticmethod  
def get_order_details_by_order(OrderID):  
    try:  
        conn = DBConnections.get_connection()  
        cursor = conn.cursor(dictionary=True)  
  
        # Get basic order details with product info  
        query = """  
            SELECT od.*, p.ProductName, p.Description, p.Price  
            FROM OrderDetails od  
            JOIN Products p ON od.ProductID = p.ProductID  
            WHERE od.OrderID = %s  
            """  
        cursor.execute(query, (OrderID,))  
  
        # Get the order object first (simple version)  
        order = Orders(OrderID=OrderID, Customer=None) # Customer can be  
        None if not needed  
  
        return [
```

```

OrderDetails(
    OrderDetailID = row['OrderDetailID'],
    Order = order, # Now passing Order object instead of just ID
    Product = Product(
        ProductID = row['ProductID'],
        ProductName = row['ProductName'],
        Description = row['Description'], # No longer empty string
        Price = row['Price']
    ),
    Quantity=row['Quantity'],
    Discount=row.get('Discount', 0.0) # Added discount with default
)
for row in cursor.fetchall()
]

except mysql.connector.Error as e:
    raise DatabaseError(f"Database error: {e}")
finally:
    if 'cursor' in locals():
        cursor.close()

inventory_dao.py:
from util.DB_Connections import DBConnections
from entity.inventory import Inventory
from entity.product import Product
from exception.exceptionHandling import ProductNotFoundError,
InvalidDataError, DatabaseError
import mysql.connector

```

```
class InventoryDAO():

    @staticmethod
    def get_inventory(ProductID):
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary = True)

            query = """
                SELECT Inv.*, P.ProductName, P.Price
                FROM Inventory Inv
                LEFT JOIN Products P
                ON Inv.ProductID = P.ProductID
                WHERE Inv.ProductID = %s
                """
            cursor.execute(query,(ProductID,))
            result = cursor.fetchone()

            if not result:
                raise ProductNotFoundError (ProductID)

            return Inventory(
                InventoryID = result['InventoryID'],
                Product = Product(
                    ProductID=result['ProductID'],
                    ProductName=result['ProductName'],
                    Description="",

```

```
        Price=result['Price']),
        QuantityInStock = result['QuantityInStock'],
        LastStockUpdate = result['LastStockUpdate']
    )
except mysql.connector.Error as e:
    raise DatabaseError (f'Database Interaction Failed : {e.msg}') from e
```

```
finally:
```

```
    if 'cursor' in locals():
        cursor.close()
```

```
@staticmethod
```

```
def get_products_by_quantity(Quantity):
```

```
try:
```

```
    conn = DBConnections.get_connection()
    cursor = conn.cursor(dictionary = True)
```

```
    query = "SELECT ProductName, ProductID, QuantityInStock FROM
Inventory WHERE QuantityInStock < %s"
```

```
    cursor.execute(query, (Quantity,))
    results = cursor.fetchall()
```

```
if not result:
```

```
    raise InvalidDataError (Quantity, "No matches found")
```

```
inventory = []
```

```
for result in results:
```

```
        inventory.append(Inventory(  
            ProductName = result['ProductName'],  
            ProductID = result['ProductID'],  
            QuantityInStock = result['QuantityInStock'])  
    ))
```

except Exception as e:

```
    raise DatabaseError(f"Database Interaction failed : {e.msg}") from e
```

finally:

```
    if 'cursor' in locals():  
        cursor.close()
```

@staticmethod

```
def add_to_inventory(Product, Quantity):
```

try:

```
    conn = DBConnections.get_connection()  
    cursor = conn.cursor(dictionary = True)
```

query = """

```
    UPDATE Inventory SET QuantityInStock = QuantityInStock + %s,  
    LastStockUpdate = CURRENT_DATE
```

```
    WHERE ProductID = %s"""
```

```
cursor.execute(query,(Quantity, Product.ProductID))
```

```
conn.commit()
```

```
return cursor.lastrowid
```

```
except mysql.connector.Error as e:  
    raise DatabaseError (f'Database Interaction failed: {e.msg}') from e  
  
finally:  
    if 'cursor' in locals():  
        cursor.close()  
  
@staticmethod  
def update_inventory(inventory):  
    try:  
        conn = DBConnections.get_connection()  
        cursor = conn.cursor()  
  
        query = """  
UPDATE Inventory SET QuantityInStock = %s, LastStockUpdate = %s  
WHERE InventoryID = %s  
"""  
        cursor.execute(query, (  
            inventory.QuantityInStock,  
            inventory.LastStockUpdate,  
            inventory.InventoryID  
        ))  
        conn.commit()  
        return cursor.rowcount > 0  
  
    except mysql.connector.Error as e:  
        conn.rollback()
```

```
        raise DatabaseError(f"Failed to update inventory: {e.msg}") from e
    finally:
        if 'cursor' in locals():
            cursor.close()

    @staticmethod
    def get_low_stock_products(threshold):
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary=True)

            query = """
                SELECT p.ProductID, p.ProductName, p.Description, p.Price,
                i.QuantityInStock, i.LastStockUpdate
                FROM Inventory i
                JOIN Products p ON i.ProductID = p.ProductID
                WHERE i.QuantityInStock < %s
                ORDER BY i.QuantityInStock
            """
            cursor.execute(query, (threshold,))
            return cursor.fetchall()

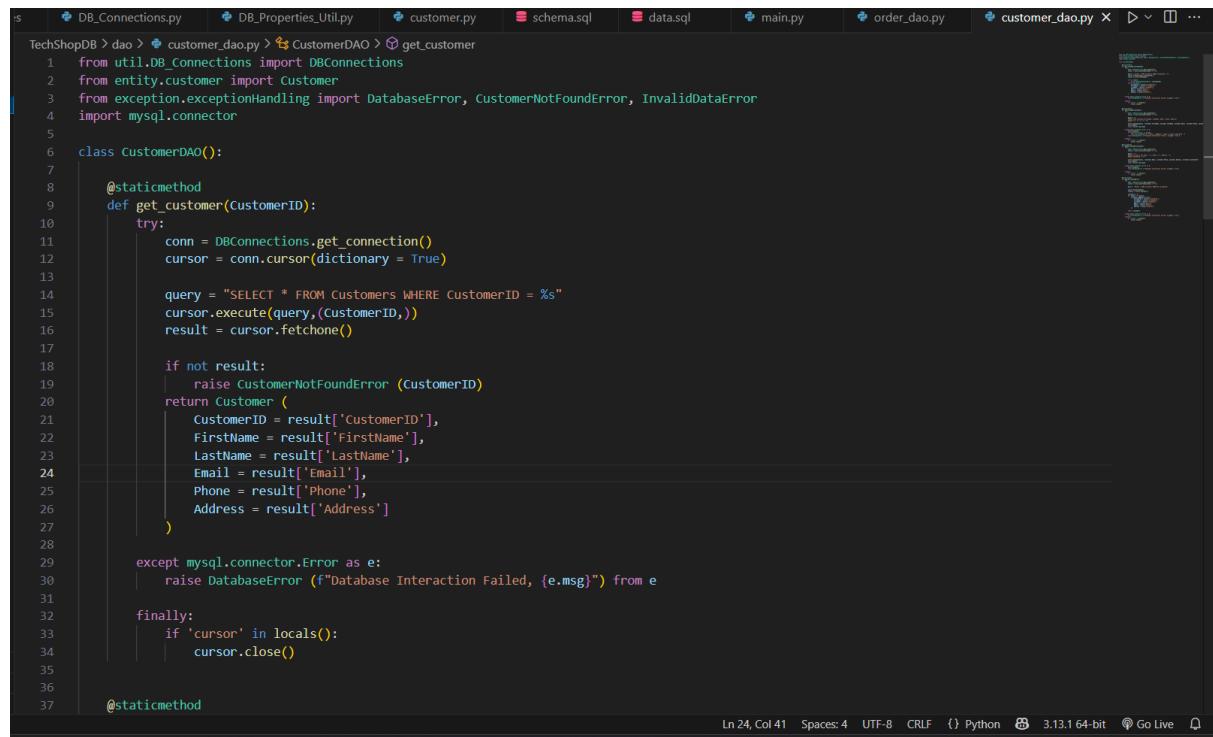
        except mysql.connector.Error as e:
            raise DatabaseError(f"Failed to get low stock products: {e.msg}") from e
        finally:
            if 'cursor' in locals():
                cursor.close()
```

```
@staticmethod
def get_out_of_stock_products():
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary=True)

        query = """
            SELECT p.ProductID, p.ProductName, p.Description, p.Price,
            i.QuantityInStock, i.LastStockUpdate
            FROM Inventory i
            JOIN Products p ON i.ProductID = p.ProductID
            WHERE i.QuantityInStock = 0
        """
        cursor.execute(query)
        return cursor.fetchall()

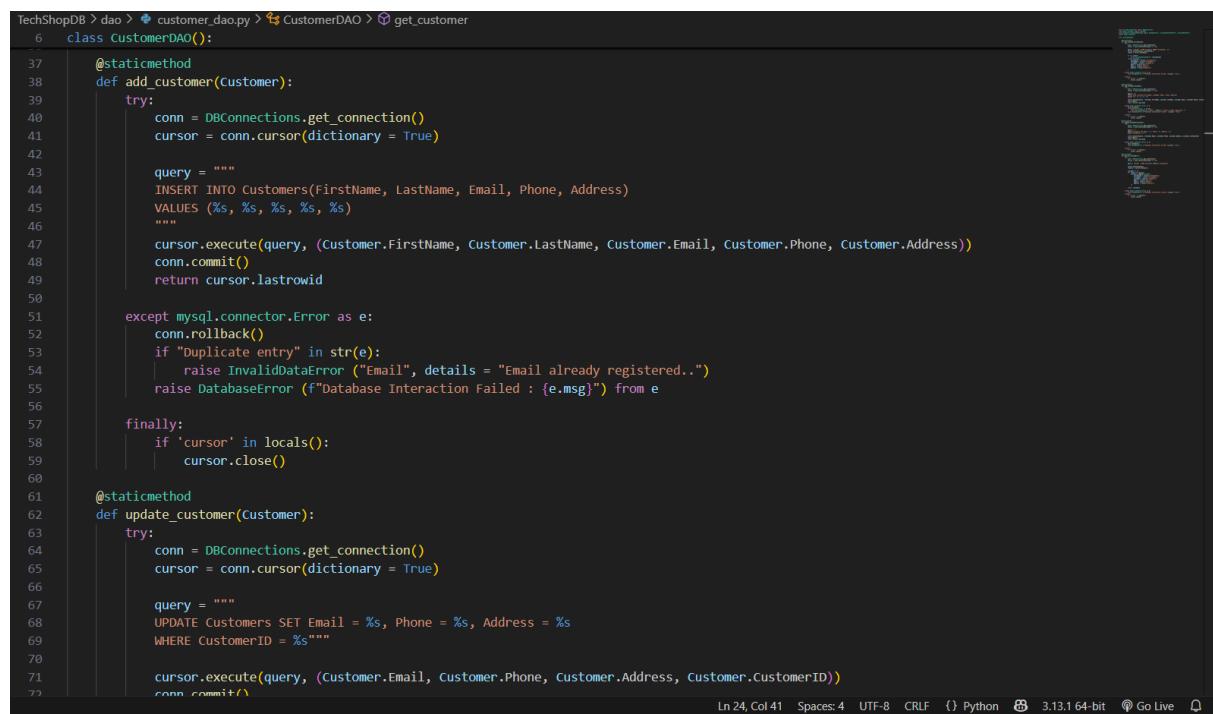
    except mysql.connector.Error as e:
        raise DatabaseError(f"Failed to get out of stock products: {e.msg}")
    finally:
        if 'cursor' in locals():
            cursor.close()
```

Output:



```
is DB_Connections.py DB_Properties_Util.py customer.py schema.sql data.sql main.py order_dao.py customer_dao.py ...
TechShopDB > dao > customer_dao.py > CustomerDAO > get_customer
1 from util.DB_Connections import DBConnections
2 from entity.customer import Customer
3 from exception.exceptionHandling import DatabaseError, CustomerNotFoundError, InvalidDataError
4 import mysql.connector
5
6 class CustomerDAO():
7
8     @staticmethod
9     def get_customer(CustomerID):
10         try:
11             conn = DBConnections.get_connection()
12             cursor = conn.cursor(dictionary = True)
13
14             query = "SELECT * FROM Customers WHERE CustomerID = %s"
15             cursor.execute(query,(CustomerID,))
16             result = cursor.fetchone()
17
18             if not result:
19                 raise CustomerNotFoundError (CustomerID)
20             return Customer (
21                 CustomerID = result['CustomerID'],
22                 FirstName = result['FirstName'],
23                 LastName = result['LastName'],
24                 Email = result['Email'],
25                 Phone = result['Phone'],
26                 Address = result['Address']
27             )
28
29         except mysql.connector.Error as e:
30             raise DatabaseError (f"Database Interaction Failed, {e.msg}") from e
31
32     finally:
33         if 'cursor' in locals():
34             cursor.close()
35
36     @staticmethod
37
```

Ln 24, Col 41 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit ⚙ Go Live



```
is DB_Connections.py DB_Properties_Util.py customer.py schema.sql data.sql main.py order_dao.py customer_dao.py ...
TechShopDB > dao > customer_dao.py > CustomerDAO > add_customer
6 class CustomerDAO():
7
8     @staticmethod
9     def add_customer(Customer):
10        try:
11            conn = DBConnections.get_connection()
12            cursor = conn.cursor(dictionary = True)
13
14            query = """
15                INSERT INTO Customers(FirstName, LastName, Email, Phone, Address)
16                VALUES (%s, %s, %s, %s, %s)
17            """
18
19            cursor.execute(query, (Customer.FirstName, Customer.LastName, Customer.Email, Customer.Phone, Customer.Address))
20            conn.commit()
21            return cursor.lastrowid
22
23        except mysql.connector.Error as e:
24            conn.rollback()
25            if "Duplicate entry" in str(e):
26                raise InvalidDataError ("Email", details = "Email already registered..")
27            raise DatabaseError (f"Database Interaction Failed : {e.msg}") from e
28
29        finally:
30            if 'cursor' in locals():
31                cursor.close()
32
33     @staticmethod
34     def update_customer(Customer):
35        try:
36            conn = DBConnections.get_connection()
37            cursor = conn.cursor(dictionary = True)
38
39            query = """
40                UPDATE Customers SET Email = %s, Phone = %s, Address = %s
41                WHERE CustomerID = %s"""
42
43            cursor.execute(query, (Customer.Email, Customer.Phone, Customer.Address, Customer.CustomerID))
44            conn.commit()
45
46
```

Ln 24, Col 41 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit ⚙ Go Live

```
DB_Connections.py DB_Properties_Util.py customer.py schema.sql data.sql main.py order_dao.py customer_dao.py

TechShopDB > dao > customer_dao.py > CustomerDAO > get_customer
6   class CustomerDAO():
62     def update_customer(customer):
69       WHERE CustomerID = %s"""
70
71       cursor.execute(query, (Customer.Email, Customer.Phone, Customer.Address, Customer.CustomerID))
72       conn.commit()
73       return cursor.lastrowid
74
75     except mysql.connector.Error as e:
76       conn.rollback()
77       raise DatabaseError (f"Database Interaction failed: {e.msg}") from e
78
79     finally:
80       if 'cursor' in locals():
81         cursor.close()
82
83   @staticmethod
84   def get_all_customers():
85     try:
86       conn = DBConnections.get_connection()
87       cursor = conn.cursor(dictionary = True)
88
89       query = "SELECT * FROM Customers ORDER BY CustomerID"
90
91       cursor.execute(query)
92       results = cursor.fetchall()
93
94       customers = []
95       for result in results:
96         customers.append(Customer(
97           CustomerID = result['CustomerID'],
98           FirstName = result['FirstName'],
99           LastName = result['LastName'],
100          Email = result['Email'],
101          Phone = result['Phone'],
102          Address = result['Address']
103        ))
104
105     return customers
106
107   except mysql.connector.Error as e:
108     raise DatabaseError (f"Database Interaction failed: {e.msg}") from e
109   finally:
110     if 'cursor' in locals():
111       cursor.close()

Ln 24, Col 41  Spaces: 4  UTF-8  CRLF  {} Python  3.13.1 64-bit  Go Live
```

```
TechShopDB > dao > customer_dao.py > CustomerDAO > get_customer
6   class CustomerDAO():
62     def update_customer(customer):
69       if 'cursor' in locals():
70         cursor.close()
71
72   @staticmethod
73   def get_all_customers():
74     try:
75       conn = DBConnections.get_connection()
76       cursor = conn.cursor(dictionary = True)
77
78       query = "SELECT * FROM Customers ORDER BY CustomerID"
79
80       cursor.execute(query)
81       results = cursor.fetchall()
82
83       customers = []
84       for result in results:
85         customers.append(Customer(
86           CustomerID = result['CustomerID'],
87           FirstName = result['FirstName'],
88           LastName = result['LastName'],
89           Email = result['Email'],
90           Phone = result['Phone'],
91           Address = result['Address']
92         ))
93
94     return customers
95
96   except mysql.connector.Error as e:
97     raise DatabaseError (f"Database Interaction failed: {e.msg}") from e
98   finally:
99     if 'cursor' in locals():
100       cursor.close()
```

Product_dao.py:

```

TechShopDB > dao > product_dao.py > ProductDAO > search_products
 1  from util.DB_Connections import DBConnections
 2  from entity.product import Product
 3  from exception.exceptionHandling import DatabaseError, ProductNotFoundError, InvalidDataError
 4  import mysql.connector
 5
 6  class ProductDAO():
 7
 8      @staticmethod
 9      def get_product(ProductId):
10          try:
11              conn = DBConnections.get_connection()
12              cursor = conn.cursor(dictionary = True)
13
14              query = "SELECT * FROM Products WHERE ProductID = %s"
15              cursor.execute(query,(ProductId,))
16              result = cursor.fetchone()
17
18              if not result:
19                  raise ProductNotFoundError(ProductID)
20
21              return Product (
22                  ProductID = result['ProductID'],
23                  ProductName = result['ProductName'],
24                  Description = result['Description'],
25                  Price = result['Price']
26              )
27
28          except mysql.connector.Error as e:
29              raise DatabaseError (f"Database Interaction Failed: {e.msg}") from e
30
31          finally:
32              if 'cursor' in locals():
33                  cursor.close()
34
35      @staticmethod
36      def add_product(Product):
37          try:

```

Ln 167, Col 16 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```

TechShopDB > dao > product_dao.py > ProductDAO > search_products
 6  class ProductDAO():
 7
 8      @staticmethod
 9      def add_product(Product):
10          try:
11
12              if not Product.ProductName or not Product.Description:
13                  raise InvalidDataError("Product", details = "Product name and Description are required")
14              if Product.Price <= 0:
15                  raise InvalidDataError("Product", details = "Product Price must be positive")
16
17              conn = DBConnections.get_connection()
18              cursor = conn.cursor(dictionary = True)
19
20              conn.start_transaction()
21
22              check_query = "SELECT ProductID FROM Products WHERE ProductName = %s"
23              cursor.execute(check_query,(Product.ProductName,))
24              if cursor.fetchone():
25                  raise InvalidDataError ("Product", details = "Product already exists in the database")
26
27              product_query = """
28                  INSERT INTO Products(ProductName, Description, Price)
29                  VALUES (%s, %s, %s)
30                  """
31
32              cursor.execute(product_query, (Product.ProductName, Product.Description, Product.Price))
33              product_id = cursor.lastrowid
34
35              inventory_query = """INSERT INTO Inventory(ProductID, QuantityInStock, LastStockUpdate)
36                  VALUES (%s, 0, CURRENT_DATE)"""
37              cursor.execute(inventory_query, (product_id,))
38
39              conn.commit()
40              return product_id
41
42          except mysql.connector.Error as e:
43              if conn:
44                  conn.rollback()

```

Ln 167, Col 16 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
 6   class ProductDAO():
 7       def add_product(Product):
 8           try:
 9               conn = DBConnections.get_connection()
10               cursor = conn.cursor(dictionary = True)
11
12               query = """
13                   UPDATE Products SET ProductName = %s, Description = %s, Price = %s
14                   WHERE ProductID = %s"""
15
16               cursor.execute(query, (product.ProductName, product.Description, product.Price, product.ProductID))
17               conn.commit()
18               return cursor.rowcount > 0
19
20           except mysql.connector.Error as e:
21               conn.rollback()
22               raise DatabaseError (f"Database Interaction Failed : {e.msg}") from e
23
24           finally:
25               if 'cursor' in locals():
26                   cursor.close()
27
28   @staticmethod
29   def update_product(product_obj):
30       try:
31           conn = DBConnections.get_connection()
32           cursor = conn.cursor(dictionary = True)
33
34           query = """
35               UPDATE Products SET ProductName = %s, Description = %s, Price = %s
36               WHERE ProductID = %s"""
37
38           cursor.execute(query, (product_obj.ProductName, product_obj.Description, product_obj.Price, product_obj.ProductID))
39           conn.commit()
40           return cursor.rowcount > 0
41
42       except mysql.connector.Error as e:
43           conn.rollback()
44           raise DatabaseError (f"Database Interaction failed: {e.msg}") from e
45
46       finally:
47           if 'cursor' in locals():
48               cursor.close()
49
50   @staticmethod
```

Ln 70, Col 28 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
 6   class ProductDAO():
 7       @staticmethod
 8       def delete_product(productId):
 9           try:
10               conn = DBConnections.get_connection()
11               cursor = conn.cursor(dictionary = True)
12
13               inventory_query = "DELETE FROM Inventory WHERE ProductID = %s"
14               cursor.execute(inventory_query,(productId,))
15
16               product_query = "DELETE FROM Products WHERE ProductID = %s"
17               cursor.execute(product_query,(productId,))
18               conn.commit()
19
20               return cursor.rowcount > 0
21
22           except mysql.connector.Error as e:
23               raise DatabaseError (f"Database Interaction Failed: {e.msg}") from e
24           except Exception as e:
25               print(f"Unexpected error: {e}")
26
27           finally:
28               if 'cursor' in locals():
29                   cursor.close()
30
31   @staticmethod
32   def get_all_products():
33       try:
34           conn = DBConnections.get_connection()
35           cursor = conn.cursor(dictionary = True)
36
37           query = "SELECT * FROM Products ORDER BY ProductID"
38           cursor.execute(query)
39           results = cursor.fetchall()
40
41           products = []
42           for result in results:
```

Ln 70, Col 28 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
6     class ProductDAO():
127    def get_all_products():
136        products = []
137        for result in results:
138            products.append(Product(
139                ProductID = result['ProductID'],
140                ProductName = result['ProductName'],
141                Description = result['Description'],
142                Price = result['Price']
143            ))
144
145        return products
146
147    except mysql.connector.Error as e:
148        raise DatabaseError(f"Database Interaction Failed: {e.msg}") from e
149
150    finally:
151        if 'cursor' in locals():
152            cursor.close()
153
154
155    @staticmethod
156    def search_products(keyword=None, min_price=None, max_price=None):
157        """Search products by keyword and/or price range"""
158        try:
159            conn = DBConnections.get_connection()
160            cursor = conn.cursor(dictionary=True)
161
162            query = """
163                SELECT p.*, i.QuantityInStock
164                FROM Products p
165                LEFT JOIN Inventory i ON p.ProductID = i.ProductID
166                WHERE 1=1
167                """
168
169            params = []
170
171            # Keyword search (name or description)
```

Ln 70, Col 28 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
6     class ProductDAO():
156    def search_products(keyword=None, min_price=None, max_price=None):
157        # Keyword search (name or description)
158        if keyword:
159            query += " AND (p.ProductName LIKE %s OR p.Description LIKE %s)"
160            params.extend([f"%{keyword}%", f"%{keyword}%"})
161
162        # Price range
163        if min_price is not None:
164            query += " AND p.Price >= %s"
165            params.append(float(min_price))
166        if max_price is not None:
167            query += " AND p.Price <= %s"
168            params.append(float(max_price))
169
170        query += " ORDER BY p.ProductName"
171
172        cursor.execute(query, params)
173        results = cursor.fetchall()
174
175        products = []
176        for result in results:
177            products.append(Product(
178                ProductID=result['ProductID'],
179                ProductName=result['ProductName'],
180                Description=result['Description'],
181                Price=result['Price']
182            ))
183
184        return products
185
186    except mysql.connector.Error as e:
187        raise DatabaseError(f"Search failed: {e.msg}") from e
188    except ValueError as e:
189        raise DatabaseError(f"Invalid price value: {e}") from e
190
191    finally:
192        if 'cursor' in locals():
193            cursor.close()
```

Ln 70, Col 28 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

Order_dao.py:

```
TechShopDB > dao > order_dao.py > OrderDAO > get_order_count
 1  from util.DB_Connections import DBConnections
 2  from entity.orders import Orders
 3  from exception.exceptionHandling import OrderNotFoundError, OrdersNotPlacedError, DatabaseError
 4  import mysql.connector
 5  from datetime import date
 6
 7  class OrderDAO():
 8
 9      @staticmethod
10      def get_order(OrderID):
11
12          try:
13              conn = DBConnections.get_connection()
14              cursor = conn.cursor(dictionary = True)
15
16              query = "SELECT * FROM Orders WHERE OrderID = %s"
17              cursor.execute(query,(OrderID,))
18              result = cursor.fetchone()
19
20              if not result:
21                  raise OrderNotFoundError(OrderID)
22
23              return Orders (
24                  OrderID = result['OrderID'],
25                  Customer = result['CustomerID'],
26                  OrderDate = result['OrderDate'],
27                  TotalAmount = result['TotalAmount'],
28                  Status = result['Status']
29              )
30
31          except mysql.connector.Error as e:
32              raise DatabaseError (f"Database Interaction Failed: {e}") from e
33
34      finally:
35          if 'cursor' in locals():
36              cursor.close()
37
```

Ln 77, Col 9 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > order_dao.py > OrderDAO > get_order_count
 7  class OrderDAO():
 8
 9      @staticmethod
10      def create_order(CustomerID, TotalAmount):
11          try:
12              conn = DBConnections.get_connection()
13              cursor = conn.cursor(dictionary = True)
14
15              query = """
16                  INSERT INTO Orders(CustomerID, OrderDate, TotalAmount, Status)
17                  VALUES(%s, %s, %s, 'Pending')
18              """
19
20              cursor.execute(query, (CustomerID, date.today(), TotalAmount))
21              conn.commit()
22              return cursor.lastrowid
23
24          except mysql.connector.Error as e:
25              conn.rollback()
26              raise DatabaseError (f"Database Interaction Failed: {e}") from e
27
28      finally:
29          if 'cursor' in locals():
30              cursor.close()
31
32      @staticmethod
33      def get_order_count(CustomerID):
34          try:
35              conn = DBConnections.get_connection()
36              cursor = conn.cursor(dictionary = True)
37
38              query = "SELECT COUNT(OrderID) AS OrderCount FROM Orders WHERE CustomerID = %s"
39              cursor.execute(query,(CustomerID,))
40              count = cursor.fetchone()
41
42              if not count:
43                  raise OrdersNotPlacedError(CustomerID)
44
45          finally:
46              if 'cursor' in locals():
47                  cursor.close()
48
```

Ln 77, Col 9 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```

TechShopDB > dao > order_dao.py > OrderDAO > get_order_count
 7   class OrderDAO():
 8     def get_order_count(CustomerID):
 9       if not count:
10         raise OrdersNotPlacedError(CustomerID)
11
12       return count['OrderCount']
13
14   except mysql.connector.Error as e:
15     raise DatabaseError(f"Database Interaction Failed: {e.msg}") from e
16
17   finally:
18     if 'cursor' in locals():
19       cursor.close()
20
21
22 @staticmethod
23 def get_customer_total_spent(CustomerID):
24   try:
25     conn = DBConnections.get_connection()
26     cursor = conn.cursor(dictionary = True)
27
28     query = "SELECT SUM(TotalAmount) FROM Orders WHERE CustomerID = %s"
29     cursor.execute(query,(CustomerID,))
30     TotalAmountSpent = cursor.fetchone()
31
32     if not TotalAmountSpent:
33       return OrdersNotPlacedError(CustomerID)
34
35     return TotalAmountSpent
36
37   except mysql.connector.Error as e:
38     raise DatabaseError(f"Database Interaction Failed: {e.msg}") from e
39
40   finally:
41     if 'cursor' in locals():
42       cursor.close()
43

```

Ln 77, Col 9 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.13.1 64-bit | Go Live |

orderDetails_dao.py:

```

TechShopDB > dao > orderDetails_dao.py > OrderDetailsDAO > add_order_detail
 1 < from util.DB_Connections import DBConnections
 2   from entity.orderDetails import OrderDetails
 3   from entity.product import Product
 4   from entity.orders import Orders
 5   from exception.exceptionHandling import InsufficientStockError, DatabaseError
 6   import mysql.connector
 7
 8 < class OrderDetailsDAO():
 9
10   @staticmethod
11   def add_order_detail(OrderID, ProductID, Quantity):
12     try:
13       conn = DBConnections.get_connection()
14       cursor = conn.cursor(dictionary = True)
15
16       #fetching the available stock in the inventory
17       query = "SELECT QuantityInStock FROM Inventory WHERE ProductID = %s"
18       cursor.execute(query, (ProductID,))
19
20       stock = cursor.fetchone()
21
22       #Checking if there is sufficient amount of stock in the inventory
23       if stock < Quantity:
24         raise InsufficientStockError(ProductID, Quantity, stock)
25
26       query = """
27         INSERT INTO OrderDetails(OrderID, ProductID, Quantity)
28         VALUES(%s, %s, %s)
29         """
30
31       cursor.execute(query, (OrderID, ProductID, Quantity))
32       conn.commit()
33
34       #Updating Inventory
35       query = "UPDATE Inventory SET QuantityInStock = QuantityInStock - %s WHERE ProductID = %s"
36       cursor.execute(query, (Quantity, ProductID))
37       conn.commit()

```

Ln 15, Col 13 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.13.1 64-bit | Go Live |

```

TechShopDB > dao > orderDetails.dao.py > OrderDetailsDAO > add_order_detail
  8   class OrderDetailsDAO():
  9     def add_order_detail(OrderID, ProductID, Quantity):
 10       conn.commit()
 11
 12       except mysql.connector.Error as e:
 13         conn.rollback()
 14         raise DatabaseError(f"Database Error : {e}") from e
 15
 16       finally:
 17         if 'cursor' in locals():
 18           cursor.close()
 19
 20
 21   @staticmethod
 22   def get_order_details_by_order(OrderID):
 23     try:
 24       conn = DBConnections.get_connection()
 25       cursor = conn.cursor(dictionary=True)
 26
 27       # Get basic order details with product info
 28       query = """
 29       SELECT od.*, p.ProductName, p.Description, p.Price
 30       FROM OrderDetails od
 31       JOIN Products p ON od.ProductID = p.ProductID
 32       WHERE od.OrderID = %s
 33       """
 34
 35       cursor.execute(query, (OrderID,))
 36
 37       # Get the order object first (simple version)
 38       order = Orders(OrderID=OrderID, Customer=None) # Customer can be None if not needed
 39
 40       return [
 41         OrderDetails(
 42           OrderDetailID = row['OrderDetailID'],
 43           Order = order, # Now passing Order object instead of just ID
 44           Product = Product(
 45             ProductID = row['ProductID'],
 46             ProductName = row['ProductName'],
 47             Description = row['Description'],
 48             Price = row['Price']
 49           ),
 50           Quantity=row['Quantity'],
 51           Discount=row.get('Discount', 0.0) # Added discount with default
 52         )
 53       ]
 54
 55     except mysql.connector.Error as e:
 56       raise DatabaseError(f"Database error: {e}")
 57     finally:
 58       if 'cursor' in locals():
 59         cursor.close()

```

Ln 15, Col 13 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

Inventory_dao.py:

```
TechShopDB > dao > inventory_dao.py > InventoryDAO > get_inventory
 1  from util.DB_Connections import DBConnections
 2  from entity.inventory import Inventory
 3  from entity.product import Product
 4  from exception.exceptionHandling import ProductNotFoundError, InvalidDataError, DatabaseError
 5  import mysql.connector
 6
 7  class InventoryDAO():
 8
 9      @staticmethod
10      def get_inventory(ProductID):
11          try:
12              conn = DBConnections.get_connection()
13              cursor = conn.cursor(dictionary = True)
14
15              query = """
16                  SELECT Inv.*, P.ProductName, P.Price
17                  FROM Inventory Inv
18                  LEFT JOIN Products P
19                  ON Inv.ProductID = P.ProductID
20                  WHERE Inv.ProductID = %s
21              """
22
23              cursor.execute(query,(ProductID,))
24              result = cursor.fetchone()
25
26              if not result:
27                  raise ProductNotFoundError (ProductID)
28
29          return Inventory(
30              InventoryID = result['InventoryID'],
31              Product = Product(
32                  ProductID=result['ProductID'],
33                  ProductName=result['ProductName'],
34                  Description='',
35                  Price=result['Price']),
36                  QuantityInStock = result['QuantityInStock'],
37                  LastStockUpdate = result['LastStockUpdate']
38          )

```

Ln 27, Col 13 Spaces:4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > inventory_dao.py > InventoryDAO > get_inventory
 7  class InventoryDAO():
 8      def get_inventory(ProductID):
 9
10          except mysql.connector.Error as e:
11              raise DatabaseError (f"Database Interaction Failed : {e.msg}") from e
12
13      finally:
14          if 'cursor' in locals():
15              cursor.close()
16
17      @staticmethod
18      def get_products_by_quantity(Quantity):
19          try:
20              conn = DBConnections.get_connection()
21              cursor = conn.cursor(dictionary = True)
22
23              query = "SELECT ProductName, ProductID, QuantityInStock FROM Inventory WHERE QuantityInStock < %s"
24              cursor.execute(query, (Quantity,))
25              results = cursor.fetchall()
26
27              if not results:
28                  raise InvalidDataError (Quantity, "No matches found")
29
30              inventory = []
31              for result in results:
32                  inventory.append(Inventory(
33                      ProductName = result['ProductName'],
34                      ProductID = result['ProductID'],
35                      QuantityInStock = result['QuantityInStock']
36                  ))
37
38          except Exception as e:
39              raise DatabaseError(f"Database Interaction failed : {e.msg}") from e
40
41      finally:
42          if 'cursor' in locals():
43              cursor.close()

```

Ln 27, Col 13 Spaces:4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > inventory.dao.py > InventoryDAO > get_inventory
 7  class InventoryDAO():
 8
 9      @staticmethod
10      def add_to_inventory(Product, Quantity):
11          try:
12              conn = DBConnections.get_connection()
13              cursor = conn.cursor(dictionary = True)
14
15              query = """
16                  UPDATE Inventory SET QuantityInStock = QuantityInStock + %s, LastStockUpdate = CURRENT_DATE
17                  WHERE ProductID = %s"""
18
19              cursor.execute(query,(Quantity, Product.ProductID))
20              conn.commit()
21              return cursor.lastrowid
22
23      except mysql.connector.Error as e:
24          raise DatabaseError(f"Database Interaction failed: {e.msg}") from e
25
26      finally:
27          if 'cursor' in locals():
28              cursor.close()
29
30
31      @staticmethod
32      def update_inventory(inventory):
33          try:
34              conn = DBConnections.get_connection()
35              cursor = conn.cursor()
36
37              query = """
38                  UPDATE Inventory SET QuantityInStock = %s, LastStockUpdate = %s
39                  WHERE InventoryID = %s"""
40
41              cursor.execute(query, (
42                  inventory.QuantityInStock,
43                  inventory.LaststockUpdate,
44                  inventory.InventoryID
45              ))
46
47          except mysql.connector.Error as e:
48              conn.rollback()
49              raise DatabaseError(f"Failed to update inventory: {e.msg}") from e
50
51          finally:
52              if 'cursor' in locals():
53                  cursor.close()
54
55
56      @staticmethod
57      def get_low_stock_products(threshold):
58          try:
59              conn = DBConnections.get_connection()
60              cursor = conn.cursor(dictionary=True)
61
62              query = """
63                  SELECT p.ProductID, p.ProductName, p.Description, p.Price, i.QuantityInStock, i.LastStockUpdate
64                  FROM Inventory i
65                  JOIN Products p ON i.ProductID = p.ProductID
66                  WHERE i.QuantityInStock < %s
67                  ORDER BY i.QuantityInStock
68              """
69
70              cursor.execute(query, (threshold,))
71              return cursor.fetchall()
72
73      except mysql.connector.Error as e:
74          raise DatabaseError(f"Failed to get low stock products: {e.msg}") from e
75
76      finally:
77          if 'cursor' in locals():
78              cursor.close()
79
80
81      @staticmethod
82      def get_out_of_stock_products():
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
```

Ln 27, Col 13 Spaces:4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > inventory.dao.py > InventoryDAO > get_low_stock_products
 7  class InventoryDAO():
 8
 9      @staticmethod
10      def update_inventory(inventory):
11          try:
12              conn = DBConnections.get_connection()
13              cursor = conn.cursor()
14
15              query = """
16                  UPDATE Inventory
17                  SET QuantityInStock = %s, LastStockUpdate = %s
18                  WHERE InventoryID = %s"""
19
20              cursor.execute(query, (
21                  inventory.QuantityInStock,
22                  inventory.LaststockUpdate,
23                  inventory.InventoryID
24              ))
25
26          except mysql.connector.Error as e:
27              conn.rollback()
28              raise DatabaseError(f"Failed to update inventory: {e.msg}") from e
29
30          finally:
31              if 'cursor' in locals():
32                  cursor.close()
33
34
35      @staticmethod
36      def get_low_stock_products(threshold):
37          try:
38              conn = DBConnections.get_connection()
39              cursor = conn.cursor(dictionary=True)
40
41              query = """
42                  SELECT p.ProductID, p.ProductName, p.Description, p.Price, i.QuantityInStock, i.LastStockUpdate
43                  FROM Inventory i
44                  JOIN Products p ON i.ProductID = p.ProductID
45                  WHERE i.QuantityInStock < %s
46                  ORDER BY i.QuantityInStock
47              """
48
49              cursor.execute(query, (threshold,))
50              return cursor.fetchall()
51
52      except mysql.connector.Error as e:
53          raise DatabaseError(f"Failed to get low stock products: {e.msg}") from e
54
55      finally:
56          if 'cursor' in locals():
57              cursor.close()
58
59
60      @staticmethod
61      def get_out_of_stock_products():
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
```

Ln 27, Col 13 Spaces:4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```
TechShopDB > dao > inventory.dao.py > InventoryDAO > get_out_of_stock_products
 7  class InventoryDAO():
 8
 9      @staticmethod
10      def get_out_of_stock_products():
11          try:
12              conn = DBConnections.get_connection()
13              cursor = conn.cursor(dictionary=True)
14
15              query = """
16                  SELECT p.ProductID, p.ProductName, p.Description, p.Price, i.QuantityInStock, i.LastStockUpdate
17                  FROM Inventory i
18                  JOIN Products p ON i.ProductID = p.ProductID
19                  WHERE i.QuantityInStock = 0
20              """
21
22              cursor.execute(query)
23              return cursor.fetchall()
24
25      except mysql.connector.Error as e:
26          raise DatabaseError(f"Failed to get out of stock products: {e.msg}") from e
27
28      finally:
29          if 'cursor' in locals():
30              cursor.close()
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
```

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.

Query:

DB_Properties_Util.py:

```
import configparser  
import os  
from pathlib import Path  
import traceback
```

```
class DBPropertiesUtil():
```

```
    @staticmethod
```

```
    def get_connection_string(property_file_name):  
        try:  
            prop_path = Path(__file__).parent.parent / property_file_name  
  
            if not prop_path.exists():  
                raise FileNotFoundError(f"Config file missing at: {prop_path}")  
  
            if not os.path.exists(prop_path):
```

```
        raise FileNotFoundError (f'Database properties file  
{property_file_name} not found..')

config = configparser.ConfigParser()  
config.read(prop_path)

if 'database' not in config:  
    raise ValueError (f'Required database configuration not found in  
{property_file_name}')

db_config = config['database']  
required_keys = ['host', 'database', 'user', 'password']

for key in required_keys:  
    if key not in db_config:  
        raise ValueError (f'Missing required database property : {key}')

result = {  
    'host' : db_config['host'],  
    'database' : db_config['database'],  
    'user' : db_config['user'],  
    'password' : db_config['password'],  
    'port' : db_config.get('port','3306')  
}

return result

except Exception as e:
```

```
    print(f"DEBUG ERROR: {type(e).__name__}: {str(e)}")  
    traceback.print_exc()  
    raise
```

DB_Connections.py:

```
import mysql.connector  
from util.DB_Properties_Util import DBPropertiesUtil  
from exception.exceptionHandling import DatabaseError  
  
class DBConnections():  
    __connection = None  
  
    @staticmethod  
    def get_connection():  
        try:  
            conn_params = DBPropertiesUtil.get_connection_string('db.properties')  
            connection = mysql.connector.connect(  
                host=conn_params['host'],  
                database=conn_params['database'],  
                user=conn_params['user'],  
                password=conn_params['password'],  
                port=int(conn_params.get('port', '3306'))  
            )  
            print("MySQL Database Connection established successfully!")  
            return connection  
        except mysql.connector.Error as e:  
            raise DatabaseError(f"Connection failed: {e.msg}") from e
```

```

# @staticmethod

# def db_connection_close():

    # if DBConnections.__connection and
DBConnections.__connection.is_connected():

        # DBConnections.__connection.close()

        # DBConnections.__connection = None

        # print("MySQL Database Connection has been closed..")

```

Output:

TechShopDB > util > DB_Properties_Util.py > DBPropertiesUtil > get_connection_string

```

1  import configparser
2  import os
3  from pathlib import Path
4  import traceback
5
6  class DBPropertiesutil():
7
8      @staticmethod
9      def get_connection_string(property_file_name):
10         try:
11             prop_path = Path(__file__).parent.parent / property_file_name
12
13             if not prop_path.exists():
14                 raise FileNotFoundError(f"Config file missing at: {prop_path}")
15
16             if not os.path.exists(prop_path):
17                 raise FileNotFoundError(f"Database properties file {property_file_name} not found..")
18
19             config = configparser.ConfigParser()
20             config.read(prop_path)
21
22             if 'database' not in config:
23                 raise ValueError ("Required database configuration not found in [prberty_file_name]")
24
25             db_config = config['database']
26             required_keys = ['host', 'database', 'user', 'password']
27
28             for key in required_keys:
29                 if key not in db_config:
30                     raise ValueError (f"Missing required database property : {key}")
31
32             result = {
33                 'host' : db_config['host'],
34                 'database' : db_config['database'],
35                 'user' : db_config['user'],
36                 'password' : db_config['password'],
37                 'port' : db_config.get('port', '3306')
38             }
39
40             return result
41
42         except Exception as e:
43             print(f"DEBUG ERROR: {type(e).__name__}: {str(e)}")
44             traceback.print_exc()
45             raise

```

Ln 23, Col 85 Spaces: 4 UTF-8 CRLF {} Python Go Live

```

31
32         result = {
33             'host' : db_config['host'],
34             'database' : db_config['database'],
35             'user' : db_config['user'],
36             'password' : db_config['password'],
37             'port' : db_config.get('port', '3306')
38         }
39
40         return result
41
42     except Exception as e:
43         print(f"DEBUG ERROR: {type(e).__name__}: {str(e)}")
44         traceback.print_exc()
45         raise

```

```

TechShopDB > util > DB_Connections.py > DBConnections > get_connection

1 import mysql.connector
2 from util.DB_Properties_Util import DBPropertiesUtil
3 from exception.exceptionHandling import DatabaseError
4
5
6 class DBConnections():
7     _connection = None
8
9     @staticmethod
10    def get_connection():
11        try:
12            conn_params = DBPropertiesUtil.get_connection_string('db.properties')
13            connection = mysql.connector.connect(
14                host=conn_params['host'],
15                database=conn_params['database'],
16                user=conn_params['user'],
17                password=conn_params['password'],
18                port=int(conn_params.get('port', '3306')))
19        except mysql.connector.Error as e:
20            print("MySQL Database Connection established successfully!")
21            return connection
22        except DatabaseError as e:
23            raise DatabaseError(f"Connection failed: {e.msg}")

```

1: Customer Registration:

Query:

```

if choice == 1:
    try:
        print("\n                                  Adding a new customer:")
        firstName = input("                          First Name: ")
        lastName = input("                          Last Name: ")
        email = input("                          Email: ")
        phone = input("                          Contact Number: ")
        address = input("                          Address: ")

        new_customer_details = Customer([
            CustomerID = None,
            FirstName = firstName,
            LastName = lastName,
            Email = email,
            Phone = phone,
            Address = address
        ])

        CustomerID = CustomerDAO.add_customer(new_customer_details)
        print(f"\n                                  Customer added to the database successfully! ID : {CustomerID}")

    except InvalidDataError as e:
        print(f"Validation Failed: {e}")
        continue
    except DatabaseError as e:
        print(f"Database Error: {e}")
        continue
    except Exception as e:
        print(f"Unexpected error: {e}")
        continue

```

```

@staticmethod
def add_customer(Customer):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        query = """
        INSERT INTO Customers(FirstName, LastName, Email, Phone, Address)
        VALUES (%s, %s, %s, %s, %s)
        """
        cursor.execute(query, (Customer.FirstName, Customer.LastName, Customer.Email, Customer.Phone, Customer.Address))
        conn.commit()
        return cursor.lastrowid

    except mysql.connector.Error as e:
        conn.rollback()
        if "Duplicate entry" in str(e):
            raise InvalidDataError ("Email", details = "Email already registered..")
        raise DatabaseError (f"Database Interaction Failed : {e.msg}") from e

    finally:
        if 'cursor' in locals():
            cursor.close()

```

Output:

The screenshot shows a terminal window with the following session:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - OOps\TechshopDB> python main.py
      Welcome to Techshop Management System
      1. Customer Operations
      2. Product Operations
      3. Order Operations
      4. Inventory Operations
      5. Payment Operations
      6. Exit
Enter the number of the section that you wish to browse(1 - 6): 1

      Customer Operations
      1. Add a new customer
      2. View Customer Details
      3. Update customer information
      4. View all customers
      5. Back to Main Menu
Enter your choice(1 - 5):1

      Adding a new customer:
First Name: Caren
Last Name: Smith
Email: caren2@gmail.com
Contact Number: 9789403245
Address: 99, Old Oak Road, Mumbai

      Customer added to the database successfully! ID : 12

      Customer Operations
      1. Add a new customer
      2. View Customer Details
      3. Update customer information
      4. View all customers
      5. Back to Main Menu
Enter your choice(1 - 5):2

      Viewing Customer Details:
Enter Customer ID: 12

      Customer ID: 12
      First Name: Caren

      5. Back to Main Menu
Enter your choice(1 - 5):2

      Viewing Customer Details:
Enter Customer ID: 12

      Customer ID: 12
      First Name: Caren
      Last Name: Smith
      Email: caren2@gmail.com
      Contact Number: 9789403245
      Residential Address: 99, Old Oak Road, Mumbai
      Total Number of Orders: 0

      Customer Operations
      1. Add a new customer
      2. View Customer Details
      3. Update customer information
      4. View all customers
      5. Back to Main Menu
Enter your choice(1 - 5):■

```

The terminal shows the execution of `main.py`. It first displays the main menu with option 1 selected for Customer Operations. Under Customer Operations, option 1 is selected for adding a new customer. The user enters customer details: First Name: Caren, Last Name: Smith, Email: caren2@gmail.com, Contact Number: 9789403245, and Address: 99, Old Oak Road, Mumbai. A success message indicates the customer was added with ID 12. The user then selects option 2 for viewing customer details, entering Customer ID 12. The terminal displays the customer's details: Customer ID 12, First Name Caren, Last Name Smith, Email caren2@gmail.com, Contact Number 9789403245, Residential Address 99, Old Oak Road, Mumbai, and Total Number of Orders 0. Finally, the user exits the program by selecting option 5 (Back to Main Menu) twice.

2. Product Catalog Management:

Query:

```
def UpdateProductInfo(self, ProductName = None, Price = None, Description = None):
    if ProductName:
        self.ProductName = ProductName
    if Description:
        self.Description = Description
    if Price is not None:
        self.Price = Price
```

```
@staticmethod
def update_product(product_obj):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        query = """
        UPDATE Products SET ProductName = %s, Description = %s, Price = %s
        WHERE ProductID = %s"""
        cursor.execute(query, (product_obj.ProductName, product_obj.Description, product_obj.Price, product_obj.ProductID))
        conn.commit()
        return cursor.rowcount > 0

    except mysql.connector.Error as e:
        conn.rollback()
        raise DatabaseError (f"Database Interaction failed: {e.msg}") from e

    finally:
        if 'cursor' in locals():
            cursor.close()
```

Output:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 2
Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu
Enter your choice(1 - 7):3
Updating Product Info:
Enter the Product ID: 4
Current Product Details:
Product ID: 4
Product Name: realme Narzo 50
Description: 4GB RAM, 64GB Storage
Price: Rs.10999

Enter the details to be updated (Leave it blank if you don't wish to update):
Enter Product Name [realme Narzo 50]: Redmi Note 4
Enter Product Description [4GB RAM, 64GB Storage]: 4GB RAM, 128GB Storage
Enter Product Price [10999]: 14999

Product Updated successfully!
Product Operations:
1. Add a new product
2. View product details
3. Update product info
```

```
Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu
Enter your choice(1 - 7):2
Viewing Product Details:
Enter the ProductID: 4
Product ID: 4
Product Name: Redmi Note 4
Description: 4GB RAM, 128GB Storage
Price: Rs.14999

Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu
Enter your choice(1 - 7):
```

3: Placing Customer Orders:

Query:

```
def order_operations():
    if choice == 1:
        try:
            print("Creating a new order: ")
            customerID = int(input("Enter the Customer ID: "))
            customer = CustomerDAO.get_customer(customerID)

            products = []
            while True:
                productID = int(input("Enter the ProductID [Type 0 to finish the order]: "))
                if productID == 0:
                    break
                quantity = int(input("the quantity: "))

                #Checking if it is available in the inventory
                inventory = InventoryDAO.get_inventory(productID)
                if not inventory.IsProductAvailable(quantity):
                    print(f"Only {inventory.QuantityInStock} units are available")
                    continue

                product = ProductDAO.get_product(productID)
                products.append((product, quantity))

            if not products:
                print("No products added to the cart")
                continue

            total = sum(p[0].Price * p[1] for p in products)

            orderID = OrderDAO.create_order(customerID, total)

            for product, quantity in products:
                OrderDetailsDAO.add_order_detail(orderID, product.ProductID, quantity)

        except CustomerNotFoundError as e:
            print(f"Customer not found: {e}")
            continue
```

```
376
377             total = sum(p[0].Price * p[1] for p in products)
378
379             orderID = OrderDAO.create_order(customerID, total)
380
381             for product, quantity in products:
382                 OrderDetailsDAO.add_order_detail(orderID, product.ProductID, quantity)
383
384         except CustomerNotFoundError as e:
385             print(f"Customer not found: {e}")
386             continue
387         except ValueError as e:
388             print(f"Enter valid numeric values: {e}")
389             continue
390         except DatabaseError as e:
391             print(f"Database Error: {e}")
392             continue
```

```
def IsProductAvailable(self, quantityToCheck):  
    if quantityToCheck < 0:  
        raise ValueError ("Enter a valid quantity..")  
    return self.__QuantityInStock >= quantityToCheck
```

```

class OrderDAO():

    @staticmethod
    def get_order(OrderID):

        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary = True)

            query = "SELECT * FROM Orders WHERE OrderID = %s"
            cursor.execute(query, (OrderID,))
            result = cursor.fetchone()

            if not result:
                raise OrderNotFoundError(OrderID)

            return Orders (
                OrderID = result['OrderID'],
                Customer = result['CustomerID'],
                OrderDate = result['OrderDate'],
                TotalAmount = result['TotalAmount'],
                Status = result['Status']
            )

        except mysql.connector.Error as e:
            raise DatabaseError (f"Database Interaction Failed: {e}") from e

        finally:
            if 'cursor' in locals():
                cursor.close()

```

```

@staticmethod
def add_order_detail(OrderID, ProductID, Quantity):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        #Fetching the available stock in the inventory
        query = "SELECT QuantityInStock FROM Inventory WHERE ProductID = %s"
        cursor.execute(query, (ProductID,))

        stock = cursor.fetchone()

        #Checking if there is sufficient amount of stock in the inventory
        if stock < Quantity:
            raise InsufficientStockError(ProductID, Quantity, stock)

        query = """
        INSERT INTO OrderDetails(OrderID, ProductID, Quantity)
        VALUES(%s, %s, %s)
        """

        cursor.execute(query, (OrderID, ProductID, Quantity))
        conn.commit()

        #Updating Inventory
        query = "UPDATE Inventory SET QuantityInStock = QuantityInStock - %s WHERE ProductID = %s"
        cursor.execute(query, (Quantity, ProductID))
        conn.commit()

    except mysql.connector.Error as e:
        conn.rollback()
        raise DatabaseError (f"Database Error : {e}") from e

    finally:
        if 'cursor' in locals():
            cursor.close()

```

Ln 15, Col 13 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit

Output:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 2
Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu
Enter your choice(1 - 7):5
Product Database:
Product ID: 1
Product Name: Redmi Note 11
Description: 6GB RAM, 128GB Storage
Price: Rs.14999
-----
Product ID: 2
Product Name: OnePlus Nord CE 2
Description: 8GB RAM, 128GB Storage
Price: Rs.23999
-----
Product ID: 3
Product Name: Samsung Galaxy M33
Description: 6GB RAM, 128GB Storage, 5G
Price: Rs.18999
-----
Product ID: 4
Product Name: Redmi Note 4
Description: 4GB RAM, 128GB Storage
Price: Rs.14999
```

Ln 400, Col 43 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit

```
-----
Product ID: 5
Product Name: iPhone 13
Description: 128GB Storage
Price: Rs.69900
-----
Product ID: 6
Product Name: HP Pavilion Laptop
Description: i5 11th Gen, 16GB RAM, 512GB SSD
Price: Rs.64990
-----
Product ID: 7
Product Name: Dell Inspiron Desktop
Description: i3 10th Gen, 8GB RAM, 1TB HDD
Price: Rs.32990
-----
Product ID: 8
Product Name: Boat Airdopes 441
Description: Wireless Earbuds, 40hrs Playback
Price: Rs.1999
-----
Product ID: 9
Product Name: Sony WH-CH510
Description: Wireless Headphones, 35hrs Battery
Price: Rs.3490
-----
Product ID: 10
Product Name: Canon EOS 1500D
Description: 24.1MP DSLR Camera with 18-55mm Lens
Price: Rs.32990
```

```
Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu
```

Enter your choice(1 - 7):7

powershell
powershell
powershell
powershell
python Tech...
python Tech...
powershell...
python Tech...
powershell...
python Tech...
python Tech...
python Tech...
python Tech...

Ln 400, Col 43 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- 1. Add a new product
- 2. View product details
- 3. Update product info
- 4. Delete a product
- 5. View all products
- 6. Search products
- 7. Back to Main Menu

Enter your choice(1 - 7): 7

- 1. Customer Operations
- 2. Product Operations
- 3. Order Operations
- 4. Inventory Operations
- 5. Payment Operations
- 6. Exit

Enter the number of the section that you wish to browse(1 - 6): 3

Order Operations:

- 1. Create new order
- 2. View order details
- 3. Update order status
- 4. Cancel order
- 5. View all orders
- 6. Process payment
- 7. Back to Main Menu

Enter your choice(1 - 7): 1

Creating a new order:

Enter the Customer ID: 4

Enter the ProductID [Type 0 to finish the order]: 6

Enter the quantity: 1

Enter the ProductID [Type 0 to finish the order]: 0

Order created successfully! ID: 13

Order Operations:

- 1. Create new order
- 2. View order details
- 3. Update order status
- 4. Cancel order
- 5. View all orders
- 6. Process payment
- 7. Back to Main Menu

Enter your choice(1 - 7):

+ ... ×

powershell
powershell
powershell
python Tech...
python Tech...
powershell...
powershell...
python Tech...
powershell...
python Tech...
python Tech...
python Tech...
python Tech...

4: Tracking Order Status

Query:

```
elif choice == 2:
    try:
        print("\n                  Viewing order details: ")
        orderID = int(input("                  Enter the Order ID: "))
        order = OrderDAO.get_order(orderID)

        print("\n                  Order Details: ")
        print(f"                  Order ID : {order.OrderID}")
        print(f"                  Customer ID : {order.Customer.CustomerID}")
        print(f"                  Customer : {order.Customer.FirstName} {order.Customer.LastName}")
        print(f"                  Order Date : {order.OrderDate}")
        print(f"                  Total Amount : {order.TotalAmount}")
        print(f"                  Status : {order.Status}")

        orderDetails = OrderDetailsDAO.get_order_details_by_order(orderID)
        for orderDetail in orderDetails:
            print(f"                  Product Name: {orderDetail.Product.ProductName} Quantity: {orderDetail.Quantity} Price: {orderDetail.Product.Price:.2f}")

    except OrderNotFoundError as e:
        print(f"Order not found: {e}")
        continue
    except ValueError as e:
        print(f"Enter a valid Order ID: {e}")
        continue
    except DatabaseError as e:
        print(f"Database Error: {e}")
        continue
```

```

@staticmethod
def get_order_details_by_order(OrderID):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary=True)

        # Get basic order details with product info
        query = """
SELECT od.*, p.ProductName, p.Description, p.Price
FROM OrderDetails od
JOIN Products p ON od.ProductID = p.ProductID
WHERE od.OrderID = %s
"""
        cursor.execute(query, (OrderID,))

        # Get the order object first (simple version)
        order = Orders(OrderID=OrderID, Customer=None) # Customer can be None if not needed

        return [
            OrderDetails(
                OrderDetailID = row['OrderDetailID'],
                Order = order, # Now passing Order object instead of just ID
                Product = Product(
                    ProductID = row['ProductID'],
                    ProductName = row['ProductName'],
                    Description = row['Description'], # No longer empty string
                    Price = row['Price']
                ),
                Quantity=row['Quantity'],
                Discount=row.get('Discount', 0.0) # Added discount with default
            )
            for row in cursor.fetchall()
        ]
    
```

```

except mysql.connector.Error as e:
    raise DatabaseError(f"Database error: {e}")
finally:
    if 'cursor' in locals():
        cursor.close()

```

Output:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 3

Order Operations:
1. Create new order
2. View order details
3. Update order status
4. Cancel order
5. View all orders
6. Process payment
7. Back to Main Menu
Enter your choice(1 - 7): 2

Viewing order details:
Enter the order ID: 4

Order Details:
Order ID : 4
Customer ID : 5
Customer : Vikram Joshi
Order Date : 2023-05-08
Total Amount : 69900
Status : Pending
Product Name: iPhone 13
Quantity: 1
Price: 69900.00

Order Operations:
1. Create new order
2. View order details
3. Update order status
4. Cancel order
5. View all orders
6. Process payment

```

5. Inventory Management

Query:

```
if choice == 1:
    try:
        print("\n")
        name = input("Adding a new product: ")
        Product Name: ").strip()
        description = input("Description: ").strip()
        Description: ")
        price = float(input("Price: "))
        Price: ")
        stock_quantity = int(input("Initial Stock Quantity: "))

        new_Product = Product(
            ProductID = None,
            ProductName = name,
            Description = description,
            Price = price
        )

        productID = ProductDAO.add_product(new_Product)
        inventory = InventoryDAO.get_inventory(productID)
        inventory.AddToInventory(stock_quantity)
        InventoryDAO.update_inventory(inventory)

        print(f"Product created successfully! ID: {productID}")

    except InvalidDataError as e:
        print(f"Validation error: {e}")
        continue
    except DatabaseError as e:
        print(f"Database error: {e}")
        continue
    except Exception as e:
        print(f"Unexpected Error: {e}")
        continue
```

```
@staticmethod
def add_product(Product):
    try:

        if not Product.ProductName or not Product.Description:
            raise InvalidDataError("Product", details = "Product name and Description are required")
        if Product.Price <= 0:
            raise InvalidDataError("Product", details = "Product Price must be positive")

        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        conn.start_transaction()

        check_query = "SELECT ProductID FROM Products WHERE ProductName = %s"
        cursor.execute(check_query, (Product.ProductName,))
        if cursor.fetchone():
            raise InvalidDataError ("Product", details = "Product already exists in the database")

        product_query = """
        INSERT INTO Products(ProductName, Description, Price)
        VALUES (%s, %s, %s)
        """
        cursor.execute(product_query, (Product.ProductName, Product.Description, Product.Price))
        product_id = cursor.lastrowid

        today = date.today()

        inventory_query = """INSERT INTO Inventory(ProductID, QuantityInStock, LastStockUpdate)
        VALUES (%s, 0, %s)
        """
        cursor.execute(inventory_query, (product_id,today))

        conn.commit()
        return product_id
```

```
def AddToInventory(self, quantity):
    if quantity <= 0:
        raise ValueError ("Quantity to be added must be larger than 0..")
    self.__QuantityInStock += quantity
    today = date.today()
    self.__LastStockUpdate = today
```

```
@staticmethod
def update_inventory(inventory):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor()

        query = """
UPDATE Inventory SET QuantityInStock = %s, LastStockUpdate = CURRENT_DATE
WHERE InventoryID = %s
"""
        cursor.execute(query, (
            inventory.QuantityInStock,
            inventory.InventoryID
        ))
        conn.commit()
        return cursor.rowcount > 0

    except mysql.connector.Error as e:
        conn.rollback()
        raise DatabaseError(f"Failed to update inventory: {e.msg}") from e
    finally:
        if 'cursor' in locals():
            cursor.close()
```

```
elif choice == 4:
    try:
        print("\n                                Deleting a Product: ")
        productID = int(input("                                Enter the ID of the Product to be deleted: "))

        product = ProductDAO.get_product(productID)
        print("\nProduct to be deleted: ")
        print(product.GetProductDetails())

        confirm = input("Are you sure you want to delete this product(Y/N): ").lower()
        if confirm != 'y':
            print("                                Product Deletion has been cancelled")
            return
        elif confirm != 'y' and confirm != 'n':
            print("                                Invalid input! Please try again..")

        if ProductDAO.delete_product(productID):
            print(f"                                Product {productID} has been deleted successfully.")
        else:
            print("                                Product not found!")

    except DatabaseError as e:
        print(f"Database Error: {e}")
        continue
    except ValueError as e:
        print(f"Enter a valid Product ID: {e}")
        continue
```

```

elif choice == 2:
    try:
        print("\n                  Updating Stock Quantity: ")
        product_id = int(input("                  Enter the Product ID: "))
        inventory = InventoryDAO.get_inventory(product_id)

        print(f"                  Current Stock Quantity for Product {product_id} in Inventory: {inventory.QuantityInStock}")

        updated_stock = int(input("                  Enter the updated stock quantity of Product {product_id}: "))
        if updated_stock < 0:
            raise InvalidDataError("Stock", "Stock quantity cannot be less than 0")

        inventory.QuantityInStock = updated_stock
        if InventoryDAO.update_inventory(inventory):
            print("                  Stock quantity updated successfully!")
        else:
            print("                  Failed to update stock quantity")

    except ValueError:
        print("                  Enter a valid number")
        continue
    except (InvalidDataError, ProductNotFoundError) as e:
        print(f"Error: {e}")
        continue

```

Output:

Adding New Products:

```

PS D:\Viktus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 2

Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu

Enter your choice(1 - 7):1
Adding a new product:
Product Name: Oppo F19S
Description: Smartphone with 128GB Storage, 48MP rear camera
Price: 22999
Initial Stock Quantity: 50
Product created successfully! ID: 15

Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu

Enter your choice(1 - 7):7
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations

```

```

3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu
Enter your choice(1 - 7):7
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 4
Inventory Operations:
1. View product inventory
2. Update stock quantity
3. List low stock items
4. List out of stock items
5. Add new stock
6. Remove stock
7. Back to Main menu
Enter your choice (1 - 7): 1
Enter Product ID to view inventory: 15
Inventory Details:
Inventory ID: 15
Product ID: 15
Product Name: Oppo F19S
QuantityInStock: 50
LastStockUpdate: 2025-06-26
Inventory Operations:
1. View product inventory
2. Update stock quantity
3. List low stock items
4. List out of stock items
5. Add new stock
6. Remove stock
7. Back to Main menu
Enter your choice (1 - 7): 1

```

Updating Stock Quantity:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 4
Inventory Operations:
1. View product inventory
2. Update stock quantity
3. List low stock items
4. List out of stock items
5. Add new stock
6. Remove stock
7. Back to Main menu
Enter your choice (1 - 7): 2
Updating Stock Quantity:
Enter the Product ID: 15
Current Stock Quantity for Product 15 in Inventory: 50
Enter the updated stock quantity of Product 15: 43
Stock quantity updated successfully!

Inventory Operations:
1. View product inventory
2. Update stock quantity
3. List low stock items
4. List out of stock items
5. Add new stock
6. Remove stock
7. Back to Main menu
Enter your choice (1 - 7): 1
Enter Product ID to view inventory: 15
Inventory Details:
Inventory ID: 15
Product ID: 15
Product Name: Oppo F19S

```

```
Inventory Details:  
Inventory ID: 15  
Product ID: 15  
Product Name: Oppo F19S  
QuantityInStock: 43  
LastStockUpdate: 2025-06-26
```

```
Inventory Operations:  
1. View product inventory  
2. Update stock quantity  
3. List low stock items  
4. List out of stock items  
5. Add new stock  
6. Remove stock  
7. Back to Main menu
```

```
Enter your choice (1 - 7):
```

Deleting a Product from the Inventory:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py  
Welcome to TechShop Management System  
1. Customer Operations  
2. Product Operations  
3. Order Operations  
4. Inventory Operations  
5. Payment Operations  
6. Exit
```

```
Enter the number of the section that you wish to browse(1 - 6): 2
```

```
Product Operations:  
1. Add a new product  
2. View product details  
3. Update product info  
4. Delete a product  
5. View all products  
6. Search products  
7. Back to Main Menu
```

```
Enter your choice(1 - 7):4
```

```
Deleting a Product:
```

```
Enter the ID of the Product to be deleted: 15
```

```
Product to be deleted:
```

```
Product ID: 15  
Product Name: Oppo F19S  
Description: Smartphone with 128GB Storage, 48MP rear camera  
Price: Rs.22999
```

```
Are you sure you want to delete this product(Y/N): y
```

```
Product 15 has been deleted successfully.
```

```
Product Operations:  
1. Add a new product  
2. View product details  
3. Update product info  
4. Delete a product  
5. View all products  
6. Search products  
7. Back to Main Menu
```

```
Enter your choice(1 - 7):2
```

```

Enter the number of the section that you wish to browse(1 - 6): 2
    Product Operations:
    1. Add a new product
    2. View product details
    3. Update product info
    4. Delete a product
    5. View all products
    6. Search products
    7. Back to Main Menu
Enter your choice(1 - 7):4

        Deleting a Product:
        Enter the ID of the Product to be deleted: 15

Product to be deleted:
Product ID: 15
Product Name: Oppo F19s
Description: Smartphone with 128GB Storage, 48MP rear camera
Price: Rs.22999
Are you sure you want to delete this product(Y/N): y
                Product 15 has been deleted successfully.

    Product Operations:
    1. Add a new product
    2. View product details
    3. Update product info
    4. Delete a product
    5. View all products
    6. Search products
    7. Back to Main Menu
Enter your choice(1 - 7):2

        Viewing Product Details:
        Enter the ProductID: 15

Product not found: Product 15 not found

    Product Operations:
    1. Add a new product
    2. View product details
    3. Update product info

```

6: Sales Reporting

Query:

```

TechShopDB > main.py > inventory_operations
555     def inventory_operations():
556         elif choice == 7:
557             try:
558                 print("\nGenerating Sales Summary Report...")
559
560                 conn = None
561                 cursor = None
562                 try:
563                     conn = DBConnections.get_connection()
564                     cursor = conn.cursor(dictionary=True)
565
566                     # Total sales query
567                     cursor.execute("""
568                         SELECT
569                             COUNT(DISTINCT OrderID) AS total_orders,
570                             SUM(TotalAmount) AS total_revenue,
571                             AVG(TotalAmount) AS avg_order_value
572                         FROM Orders
573                     """)
574                     sales_data = cursor.fetchone()
575
576                     # Top products query
577                     cursor.execute("""
578                         SELECT p.ProductName, SUM(od.Quantity) AS units_sold
579                         FROM OrderDetails od
580                         JOIN Products p ON od.ProductID = p.ProductID
581                         GROUP BY units_sold DESC
582                         ORDER BY units_sold DESC
583                         LIMIT 3
584                     """)
585                     top_products = cursor.fetchall()
586
587                     # Display report
588                     print("\n==== SALES SUMMARY ===")
589                     print(f"Total Orders: {sales_data['total_orders']}")
590                     print(f"Total Revenue: Rs.{sales_data['total_revenue']:.2f}")
591                     print(f"Avg Order Value: Rs.{sales_data['avg_order_value']:.2f}")
592
593                 except Exception as e:
594                     print(f"An error occurred: {e}")
595
596             finally:
597                 if cursor:
598                     cursor.close()
599                 if conn:
600                     conn.close()
601
602         else:
603             print("Invalid choice. Please enter a valid option (1-7).")
604
605     except KeyboardInterrupt:
606         print("\nProgram terminated by user.")


Ln 766, Col 31  Spaces: 4  UTF-8  CRLF  {} Python  Go Live  3.13.1 64-bit

```

```

748
749
750     # Display report
751     print("\n==== SALES SUMMARY ===")
752     print(f"Total Orders: {sales_data['total_orders']} ")
753     print(f"Total Revenue: Rs.{sales_data['total_revenue']:.2f}")
754     print(f"Avg Order Value: Rs.{sales_data['avg_order_value']:.2f}")
755
756     print("\nTop Selling Products:")
757     for i, product in enumerate(top_products, 1):
758         print(f"{i}. {product['ProductName']}: {product['units_sold']} units")
759
760     print("\n==== END REPORT ===")
761
762 except mysql.connector.Error as e:
763     print(f"Database error generating report: {e}")
764 except DatabaseError as e:
765     print(f"Database Error: {e}")
766 except Exception as e:
767     print(f"Unexpected Error: {e}")
768 finally:
769     if cursor:
770         cursor.close()
771     if conn and conn.is_connected():
772         conn.close()

```

Output:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Ops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 4
Inventory Operations:
1. View product inventory
2. Update stock quantity
3. List low stock items
4. List out of stock items
5. Add new stock
6. Remove stock
7. Generate Sales Report
8. Back to Main menu
Enter your choice (1 - 8): 7
Generating Sales Summary Report...
==== SALES SUMMARY ====
Total Orders: 13
Total Revenue: Rs.454,785.00
Avg Order Value: Rs.34,983.46

Top Selling Products:
1. Sony WH-CH510: 4 units
2. HP Pavilion Laptop: 3 units
3. Redmi Note 4: 1 units

==== END REPORT ====
Inventory Operations:
1. View product inventory
2. Update stock quantity
3. List low stock items
4. List out of stock items
5. Add new stock

```

Ln 753, Col 57 Spaces: 4 UTF-8 CRLF {} Python ⚙️ 3.13.1 64-bit ⚡ Go Live 🔍

7: Customer Account Updates

Query:

```

if customer_operations():

    elif choice == 3:
        try:
            print("\n                                Updating Customer Details: ")
            CustomerID = int(input("                                Enter the Customer ID whose details you wish to update: "))
            customer_details = CustomerDAO.get_customer(CustomerID)

            print(f"\n                                Current details of Customer {CustomerID}:")
            print(customer_details.GetCustomerDetails())

            print("\n                                Enter details to be updated(Leave the fields blank if there are no changes to be made): ")
            new_email = input(f"                                Email Address [{customer_details.Email}]: ") or customer_details.Email
            new_phone = input(f"                                Contact number [{customer_details.Phone}]: ") or customer_details.Phone
            new_address = input(f"                                Address [{customer_details.Address}]: ") or customer_details.Address

            customer_details.UpdateCustomerInfo(
                email = new_email,
                Phone = new_phone,
                Address =new_address
            )
            CustomerDAO.update_customer(customer_details)
            print("                                Customer Information Updated successfully!")

        except CustomerNotFoundError as e:
            print(f"Customer ID not found: {e}")
            continue
        except InvalidDataError as e:
            print(f"Validation Failed: {e}")
            continue
        except DatabaseError as e:
            print(f"Database Error: {e}")
            continue
        except Exception as e:
            print(f"Unexpected Error: {e}")
            continue

```

```

def UpdateCustomerInfo(self, email = None, Phone = None, Address = None):
    if email:
        self.Email = email
    if Phone:
        self.Phone = Phone
    if Address:
        self.Address = Address

```

```

@staticmethod
def update_customer(Customer):
    try:
        conn = DBConnections.get_connection()
        cursor = conn.cursor(dictionary = True)

        query = """
UPDATE Customers SET Email = %s, Phone = %s, Address = %
WHERE CustomerID = %s"""

        cursor.execute(query, (Customer.Email, Customer.Phone, Customer.Address, Customer.CustomerID))
        conn.commit()
        return cursor.lastrowid

    except mysql.connector.Error as e:
        conn.rollback()
        raise DatabaseError (f"Database Interaction failed: {e.msg}") from e

    finally:
        if 'cursor' in locals():
            cursor.close()

```

Output:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 1
Customer Operations
1. Add a new customer
2. View customer Details
3. Update customer information
4. View all customers
5. Back to Main Menu
Enter your choice(1 - 5):3
Updating Customer Details:
Enter the Customer ID whose details you wish to update: 11
Current details of customer 11:
Customer ID: 11
First Name: Chrystoline
Last Name: Caren
Email: caren24@gmail.com
Contact Number: 9873027800
Residential Address: 98, Old Park Rd, Trichy

Enter details to be updated(Leave the fields blank if there are no changes to be made):
Email Address [caren24@gmail.com]: caren123@gmail.com
Contact number [9873027800]: 9655700976
Address [98, Old Park Rd, Trichy]:
Customer Information Updated successfully!
Customer Operations
1. Add a new customer
2. View customer Details
3. Update customer information
4. View all customers
5. Back to Main Menu
Ln 753, Col 57  Spaces: 4  UTF-8  CRLF  {} Python  3.13.1 64-bit  Go Live  □

Customer Information Updated successfully!
Customer Operations
1. Add a new customer
2. View Customer Details
3. Update customer information
4. View all customers
5. Back to Main Menu
Enter your choice(1 - 5):2
Viewing Customer Details:
Enter Customer ID: 11
Customer ID: 11
First Name: Chrystoline
Last Name: Caren
Email: carent123@gmail.com
Contact Number: 9655700976
Residential Address: 98, Old Park Rd, Trichy
Total Number of Orders: 0

Customer Operations
1. Add a new customer
2. View Customer Details
3. Update customer information
4. View all customers
5. Back to Main Menu
Enter your choice(1 - 5):
```

Ln 753, Col 57 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live □

8: Payment Processing

Query:

```

def payment_operations():
    if choice == 1:
        try:
            payment_id = int(input("Enter Payment ID: "))
            payment = PaymentDAO.get_payment(payment_id)
            if payment:
                print(payment.GetPaymentDetails())
            else:
                print("Payment not found")
        except ValueError:
            print("Please enter a valid Payment ID")

    elif choice == 2:
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary=True)

            query = """
SELECT p.*, o.CustomerID, c.FirstName, c.LastName
FROM Payments p
JOIN Orders o ON p.OrderID = o.OrderID
JOIN Customers c ON o.CustomerID = c.CustomerID
ORDER BY p.PaymentDate DESC
"""
            cursor.execute(query)

            print("\nAll Payments:")
            print("-" * 80)
            for payment in cursor.fetchall():
                print(f"Payment ID: {payment['PaymentID']}")
                print(f"Order ID: {payment['OrderID']} (Customer: {payment['FirstName']} {payment['LastName']})")
                print(f"Date: {payment['PaymentDate']} | Method: {payment['PaymentMethod']}")
                print(f"Amount: {payment['Amount']} | Status: {payment['Status']}")
                print("-" * 80)

        except DatabaseError as e:
            print(e)

```

```

TechShopDB > entity > payment.py > Payment > GetPaymentDetails
4   class Payment:
31     @property
32     def Amount(self):
33       return self.__Amount
34
35     @property
36     def status(self):
37       return self.__Status
38
39     # Setters with validation
40     @PaymentMethod.setter
41     def PaymentMethod(self, value):
42       valid_methods = ['Credit Card', 'Debit Card', 'Net Banking', 'UPI', 'Cash']
43       if value not in valid_methods:
44         raise InvalidDataError("Payment Method", f"Must be one of: {', '.join(valid_methods)}")
45       self.__PaymentMethod = value
46
47     @Amount.setter
48     def Amount(self, value):
49       if value <= 0:
50         raise InvalidDataError("Amount", "Must be greater than 0")
51       self.__Amount = value
52
53     @Status.setter
54     def Status(self, value):
55       valid_statuses = ['Pending', 'Completed', 'Failed', 'Refunded']
56       if value not in valid_statuses:
57         raise InvalidDataError("Status", f"Must be one of: {', '.join(valid_statuses)}")
58       self.__Status = value
59
60     def GetPaymentDetails(self):
61       return f"Payment ID: {self.__PaymentID}\nOrder ID: {self.__OrderID}\n"

```

```

class PaymentDAO:

    @staticmethod
    def process_payment(order_id, payment_method, amount):
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary=True)

            # Validate payment amount against order total
            cursor.execute("SELECT TotalAmount FROM Orders WHERE OrderID = %s", (order_id,))
            order = cursor.fetchone()

            if not order:
                raise PaymentFailedError(order_id, "Order not found")

            if float(amount) != float(order['TotalAmount']):
                raise PaymentFailedError(order_id, "Payment amount doesn't match order total")

            # Process payment (in a real system, this would call a payment gateway)
            # For simulation, we'll just record it
            query = """
                INSERT INTO Payments (OrderID, PaymentMethod, Amount, Status)
                VALUES (%s, %s, %s, 'Completed')
            """
            cursor.execute(query, (order_id, payment_method, amount))
            payment_id = cursor.lastrowid

            # Update order status
            cursor.execute("UPDATE Orders SET Status = 'Processing' WHERE OrderID = %s", (order_id,))

            conn.commit()
            return payment_id

        except mysql.connector.Error as e:
            conn.rollback()
            raise DatabaseError(f"Payment processing failed: {e.msg}") from e
        finally:
            pass

```

```

    conn.rollback()
    raise DatabaseError(f"Payment processing failed: {e.msg}") from e
finally:
    if 'cursor' in locals():
        cursor.close()

    @staticmethod
    def get_payment(payment_id):
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary=True)

            query = "SELECT * FROM Payments WHERE PaymentID = %s"
            cursor.execute(query, (payment_id,))
            result = cursor.fetchone()

            if not result:
                return None

            return Payment(
                PaymentID=result['PaymentID'],
                OrderID=result['OrderID'],
                PaymentDate=result['PaymentDate'],
                PaymentMethod=result['PaymentMethod'],
                Amount=result['Amount'],
                Status=result['Status']
            )

        except mysql.connector.Error as e:
            raise DatabaseError(f"Database error: {e.msg}") from e
        finally:
            if 'cursor' in locals():
                cursor.close()

```

Output:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops> cd TechShopDB
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops> TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 5

Payment Operations:
1. View payment details
2. View all payments
3. Back to Main Menu
Enter your choice (1-3): 1
Enter Payment ID: 1
Payment ID: 1
Order ID: 1
Date: 2023-05-01 14:30:00
Method: Credit Card
Amount: 3490.00
Status: Completed

Payment Operations:
1. View payment details
2. View all payments
3. Back to Main Menu
Enter your choice (1-3):
```

9: Product Search and Recommendations

Query:

```
def product_operations():
    elif choice == 6:
        try:
            print("\nProduct Search")
            print("(Leave fields blank to skip criteria)")

            keyword = input("Search keyword (name or description): ").strip() or None
            min_price = input("Minimum price: ").strip() or None
            max_price = input("Maximum price: ").strip() or None

            min_price = float(min_price) if min_price else None
            max_price = float(max_price) if max_price else None

            products = ProductDAO.search_products(
                keyword=keyword,
                min_price=min_price,
                max_price=max_price
            )

            if not products:
                print("\nNo products found matching your criteria")
                continue

            print(f"\n' SEARCH RESULTS ':=^80")
            print(f'{ID:<5} {Name:<25} {Description:<30} {Price:>10} {Stock:>10}')
            print("-" * 80)
            for product in products:
                try:
                    inventory = InventoryDAO.get_inventory(product.ProductID)
                    stock = inventory.QuantityInStock if inventory else 0

                    desc = (product.Description[:27] + "...") if len(product.Description) > 30 else product.Description
                    print(f'{product.ProductID:<5} '
                          f'{product.ProductName[:24]:<25} '
                          f'{desc:<30} '
                          f"Rs.{product.Price:>8.2f} "
                          f'{stock:>10} ')
                except Exception as e:
                    print(f"Error: {e}")
        except KeyboardInterrupt:
            print("\nProduct Search terminated by user.")
```

```

def product_operations():
    )

    if not products:
        print("\nNo products found matching your criteria")
        continue

    print(f"\n{'-' * 80}")
    print(f"{'ID':<5} {'Name':<25} {'Description':<30} {'Price':>10} {'Stock':>10}")
    print("-" * 80)
    for product in products:
        try:
            inventory = InventoryDAO.get_inventory(product.ProductID)
            stock = inventory.QuantityInStock if inventory else 0

            desc = (product.Description[:27] + "...") if len(product.Description) > 30 else product.Description
            print(f"{product.ProductID:<5} "
                  f"{product.ProductName[:24]:<25} "
                  f"{desc:<30} "
                  f"Rs.{product.Price:>8.2f} "
                  f"{stock:>10}")

        except DatabaseError as e:
            print(f"(product.ProductID:<5} "
                  f"(product.ProductName[:24]:<25} "
                  f"(desc:<30} "
                  f"Rs.{product.Price:>8.2f} "
                  f"(N/A:>10}")
            continue
        print("=" * 80)

    except ValueError as e:
        print(f"\nError: Invalid price format. Please enter numbers only. {str(e)}")
    except DatabaseError as e:
        print(f"\nDatabase error during search: {str(e)}")
    except Exception as e:
        print(f"\nAn unexpected error occurred during search: {str(e)}")

```

Output:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Ops\TechShopDB> python main.py
Welcome to TechShop Management System
1. Customer Operations
2. Product Operations
3. Order Operations
4. Inventory Operations
5. Payment Operations
6. Exit
Enter the number of the section that you wish to browse(1 - 6): 2

Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu

Enter your choice(1 - 7):6

Product Search
(Leave fields blank to skip criteria)
Search keyword (name or description): Redmi
Minimum price: 5000
Maximum price: 30000

===== SEARCH RESULTS =====
ID      Name          Description           Price     Stock
--      --
1      Redmi Note 11   6GB RAM, 128GB Storage   Rs.14999.00    15
4      Redmi Note 4    4GB RAM, 128GB Storage   Rs.14999.00    0
=====

Product Operations:
1. Add a new product
2. View product details
3. Update product info
4. Delete a product
5. View all products
6. Search products
7. Back to Main Menu

```