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")
```

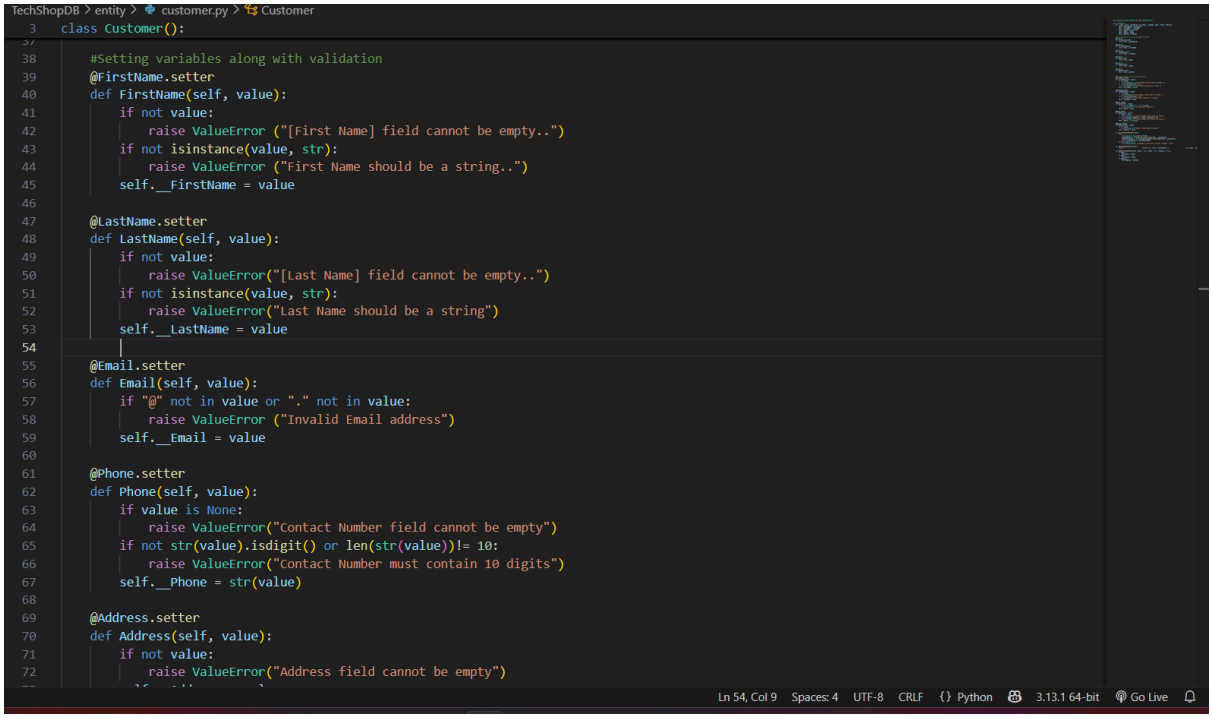
```
        self.__LastName = value

    @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)
```

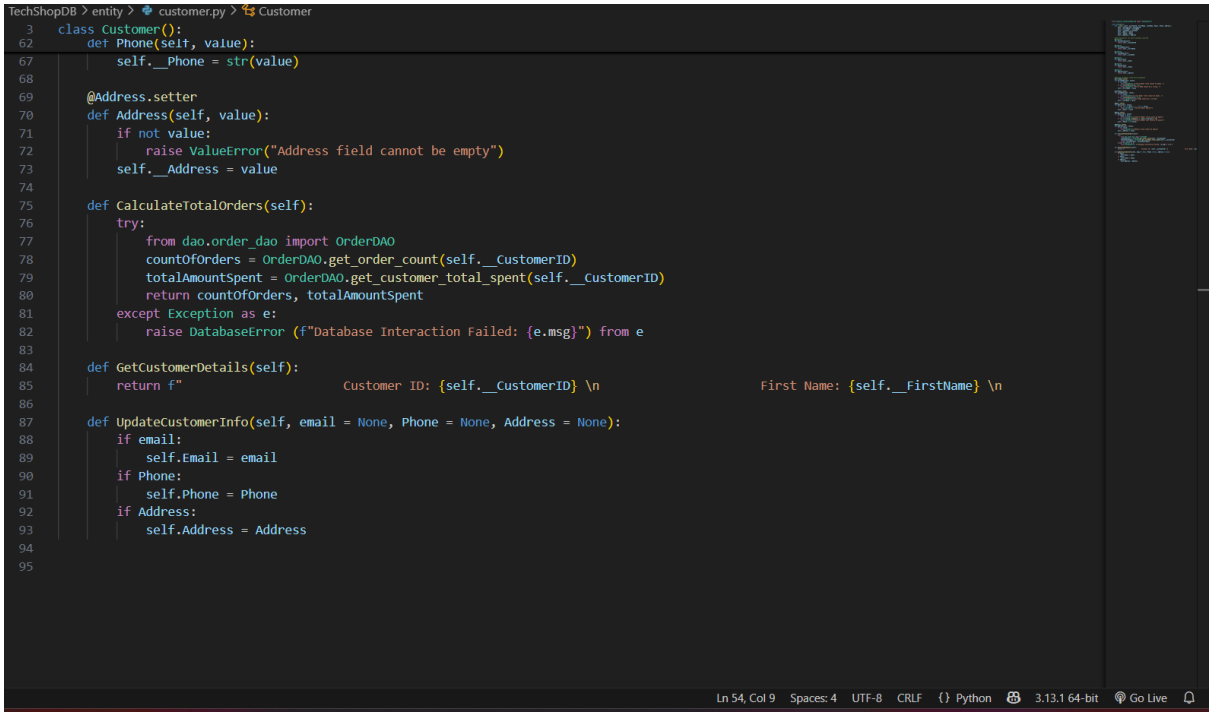
```
            return countOfOrders, totalAmountSpent
        except Exception as e:
            raise DatabaseError (f'Database Interaction Failed: {e.msg}') from e


    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:

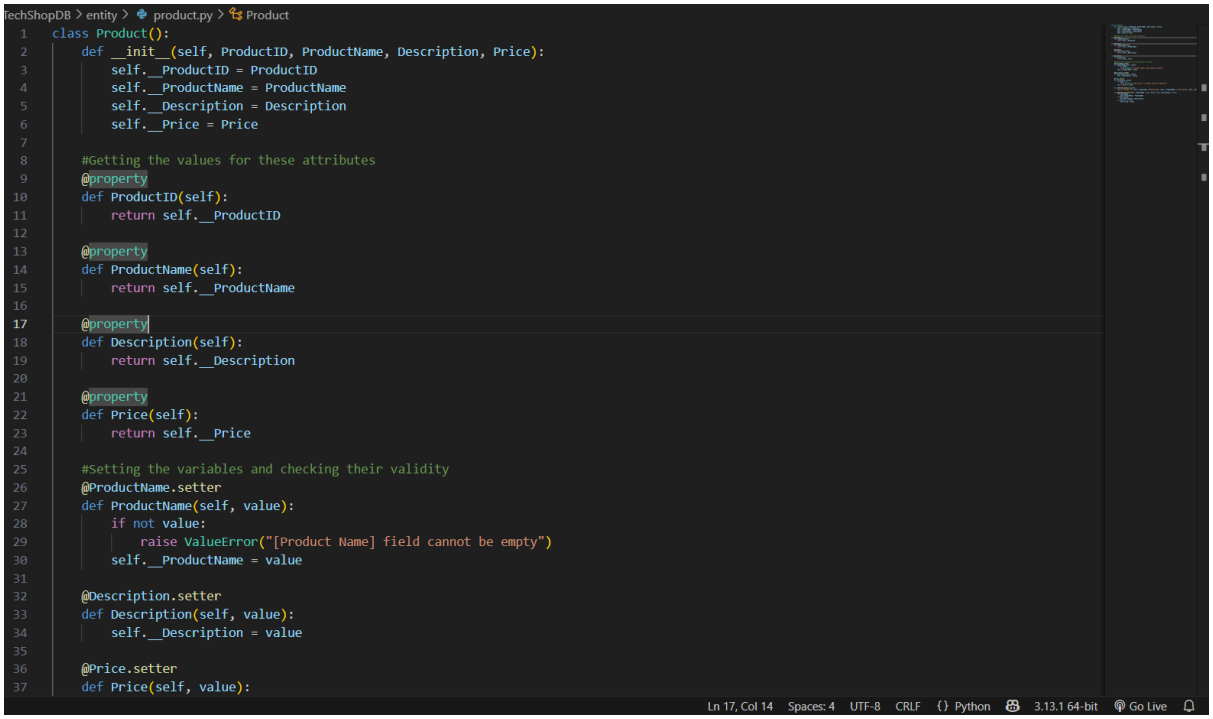
customer.py:

```python
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
class Customer():
def Phone(self, value):
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
def UpdateCustomerInfo(self, email = None, Phone = None, Address = None):
if email:
self.Email = email
if Phone:
self.Phone = Phone
if Address:
self.Address = Address
Ln 54, Col 9 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

product.py:


TechShopDB > entity > product.py > Product
class Product():
def __init__(self, ProductID, ProductName, Description, Price):
self.__ProductID = ProductID
self.__ProductName = ProductName
self.__Description = Description
self.__Price = Price
#Getting the values for these attributes
@property
def ProductID(self):
return self.__ProductID
@property
def ProductName(self):
return self.__ProductName
@property
def Description(self):
return self.__Description
@property
def Price(self):
return self.__Price
#Setting the variables and checking their validity
@ProductName.setter
def ProductName(self, value):
if not value:
raise ValueError("[Product Name] field cannot be empty")
self.__ProductName = value
@Description.setter
def Description(self, value):
self.__Description = value
@Price.setter
def Price(self, value):
Ln 17, Col 14 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```python
    @Price.setter
    def Price(self, value):
        if value < 0:
            raise ValueError("The price of a product cannot be negative")
        self.__Price = value

    def GetProductDetails(self):
        return f"Product ID: {self.__ProductID} \nProduct Name: {self.__ProductName} \n Description: {self.__Description} \nPrice: Rs.{self.__P

    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
```

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

```python
from datetime import date

class Orders():
    def __init__(self, OrderID, Customer, OrderDate = None, TotalAmount = None):
        self.__OrderID = OrderID
        self.__Customer = Customer
        self.__OrderDate = OrderDate if OrderDate else date.today()
        self.__TotalAmount = TotalAmount
        self.__Status = "Pending"

    #Getters for the attributes
    @property
    def OrderID(self):
        return self.__OrderID

    @property
    def Customer(self):
        return self.__Customer

    @property
    def OrderDate(self):
        return self.__OrderDate

    @property
    def TotalAmount(self):
        return self.__TotalAmount

    @property
    def Status(self):
        return self.__Status

    #Setters for the attributes with validity

    @Customer.setter
    def TotalAmount(self, value):
        if value < 0:
            raise ValueError ("Total Amount cannot be negative..")
```

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

```python
    @Customer.setter
    def TotalAmount(self, value):
        if value < 0:
            raise ValueError ("Total Amount cannot be negative..")
        self.__TotalAmount = value

    def CalculateTotalAmount(self):
        OrderDetails = OrderDetailsDAO.get_order_details_by_order(self.__OrderID)
        return sum(
            detail.Quantity * (detail.Product.Price * (1 - detail.Discount))
            for detail in OrderDetails
        )

    def UpdateOrderStatus(self, newStatus):
        valid_statuses = ["Pending", "Processing", "Shipped", "Delivered", "Cancelled"]
        if newStatus not in valid_statuses:
            raise ValueError (f"The status of the product should be one of these : {"," .join (valid_statuses)}")
        self.__Status = newStatus
```

orderDetails.py:

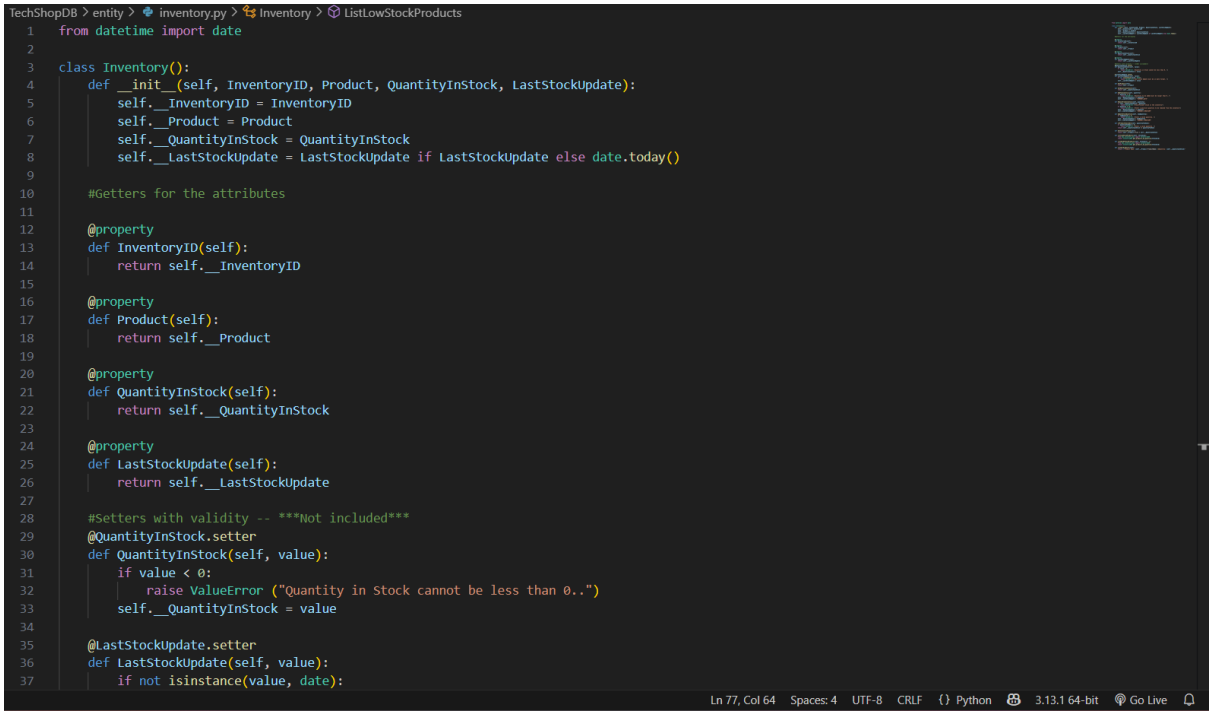
TechShopDB > entity > orderDetails.py > OrderDetails > Order

```python
class OrderDetails():
    def __init__(self, OrderDetailID, Order, Product, Quantity):
        self.__OrderDetailID = OrderDetailID
        self.__Order = Order
        self.__Product = Product
        self.__Quantity = Quantity
        self.__Discount = 0.0

    #Getters for the attributes
    @property
    def OrderDetailID(self):
        return self.__OrderDetailID

    @property
    def Order(self):
        return self.__Order

    @property
    def Product(self):
        return self.__Product

    @property
    def Quantity(self):
        return self.__Quantity

    @property
    def Discount(self):
        return self.__Discount

    #Setters for the Attributes
    @Quantity.setter
    def Quantity(self, value):
        if value <= 0:
            raise ValueError ("Quantity must be greater than 0")
        self.__Quantity = value

    # @Discount.setter
```

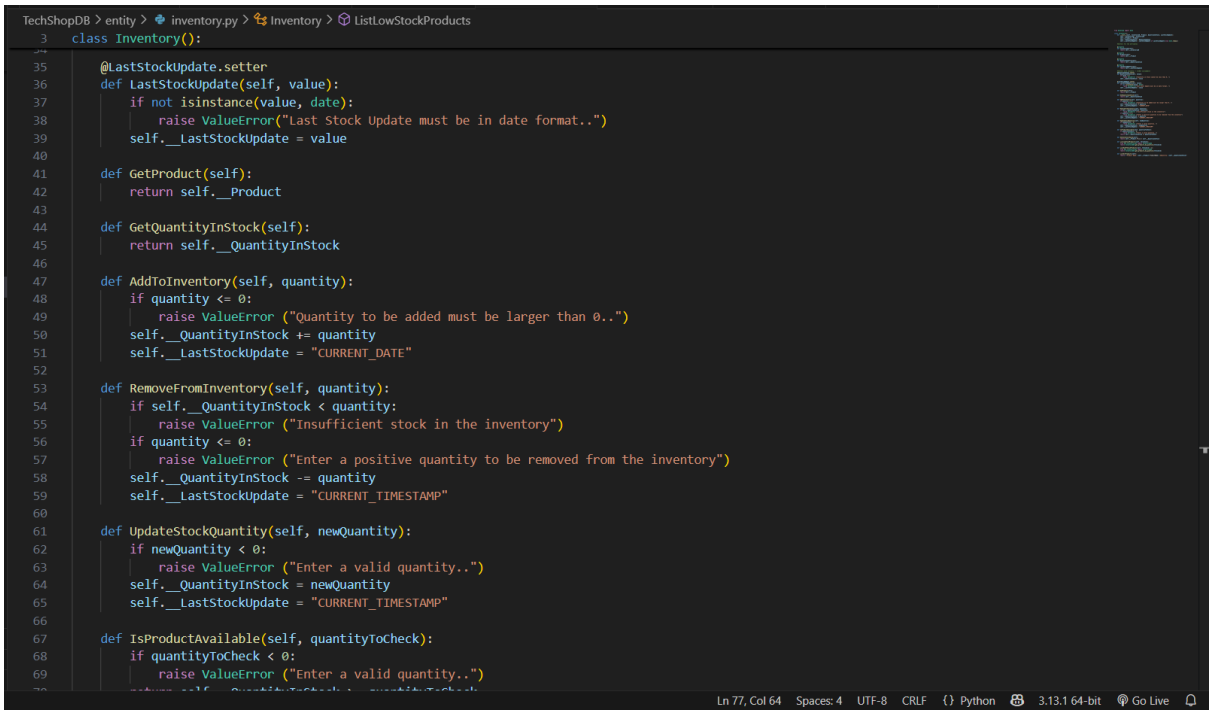
Ln 15, Col 21 Spaces: 4 UTF-8 CRLF {} Python 3.13.1 64-bit Go Live

```python
    # @Discount.setter
    # def Discount(self, value):
    #     if value < 0:
    #         raise ValueError("Discount cannot be lesser than 0")
    #     self.__Discount = value

    def CalculateSubtotal(self):
        return self.__Product.Price * self.__Quantity * (1 - self.__Discount)

    def GetOrderDetailInfo(self):
        return (f"OrderDetail ID: {self.__OrderDetailID} \nProduct Name: {self.__Product.ProductName} \nProduct Price: {self.__Product.Price:.2

    def UpdateQuantity(self, newQuantity):
        self.__Quantity = newQuantity

    def AddDiscount(self, addedDiscount):
        if not 0 <= addedDiscount <= 100:
            raise ValueError ("Discount Percentage must be in the range of 1 - 100..")
        self.__Discount += (addedDiscount/100)
```

inventory.py:



```python
from datetime import date

class Inventory():
    def __init__(self, InventoryID, Product, QuantityInStock, LastStockUpdate):
        self.__InventoryID = InventoryID
        self.__Product = Product
        self.__QuantityInStock = QuantityInStock
        self.__LastStockUpdate = LastStockUpdate if LastStockUpdate else date.today()

    #Getters for the attributes

    @property
    def InventoryID(self):
        return self.__InventoryID

    @property
    def Product(self):
        return self.__Product

    @property
    def QuantityInStock(self):
        return self.__QuantityInStock

    @property
    def LastStockUpdate(self):
        return self.__LastStockUpdate

    #Setters with validity -- ***Not included***
    @QuantityInStock.setter
    def QuantityInStock(self, value):
        if value < 0:
            raise ValueError ("Quantity in Stock cannot be less than 0..")
        self.__QuantityInStock = value

    @LastStockUpdate.setter
    def LastStockUpdate(self, value):
        if not isinstance(value, date):
```



```python
class Inventory():

    @LastStockUpdate.setter
    def LastStockUpdate(self, value):
        if not isinstance(value, date):
            raise ValueError("Last Stock Update must be in date format..")
        self.__LastStockUpdate = value

    def GetProduct(self):
        return self.__Product

    def GetQuantityInStock(self):
        return self.__QuantityInStock

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

    def RemoveFromInventory(self, quantity):
        if self.__QuantityInStock < quantity:
            raise ValueError ("Insufficient stock in the inventory")
        if quantity <= 0:
            raise ValueError ("Enter a positive quantity to be removed from the inventory")
        self.__QuantityInStock -= quantity
        self.__LastStockUpdate = "CURRENT_TIMESTAMP"

    def UpdateStockQuantity(self, newQuantity):
        if newQuantity < 0:
            raise ValueError ("Enter a valid quantity..")
        self.__QuantityInStock = newQuantity
        self.__LastStockUpdate = "CURRENT_TIMESTAMP"

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

```python
    def GetInventoryValue(self):
        return self.__Product.Price * self.__QuantityInStock

    def ListLowStockProducts(self, threshold):
        from dao.inventory_dao import InventoryDAO
        return InventoryDAO.get_products_by_quantity(threshold)

    def ListOutOfStockProducts(self, threshold = 1):
        from dao.inventory_dao import InventoryDAO
        return InventoryDAO.get_products_by_quantity(threshold)

    def ListAllProducts(self):
        return f"Product Name: {self.__Product.ProductName} \nQuantity: {self.__QuantityInStock}"
```

Task 5: Exceptions handling

• Data Validation:

o Challenge: Validate user inputs and data from external sources (e.g., user registration,

order placement).

o Scenario: When a user enters an invalid email address during registration.

o Exception Handling: Throw a custom InvalidDataException with a clear error message.

• Inventory Management:

o Challenge: Handling inventory-related issues, such as selling more products than are in

stock.

o Scenario: When processing an order with a quantity that exceeds the available stock.

o Exception Handling: Throw an InsufficientStockException and update the order status

accordingly.

• Order Processing:

o Challenge: Ensuring the order details are consistent and complete before processing.

o Scenario: When an order detail lacks a product reference.

o Exception Handling: Throw an IncompleteOrderException with a message explaining the

issue.

• Payment Processing:

o Challenge: Handling payment failures or declined transactions.

o 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

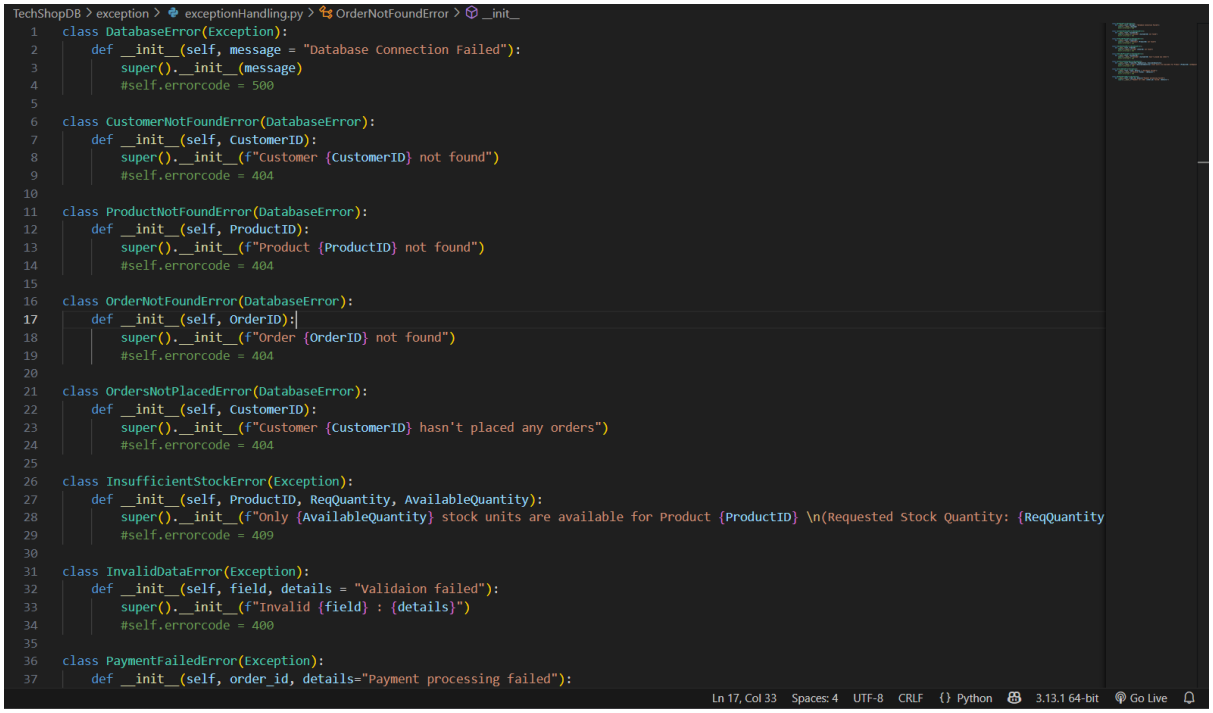
AuthorizationException to handle security-related issues.

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:



```python
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}")
```

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)
```

```
            query = "SELECT COUNT(OrderID) AS OrderCount FROM Orders
WHERE CustomerID = %s"
            cursor.execute(query,(CustomerID,))
            count = cursor.fetchone()

            if not count:
                raise OrdersNotPlacedError(CustomerID)

            return count['OrderCount']

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

            query = "SELECT SUM(TotalAmount) FROM Orders WHERE
CustomerID = %s"
            cursor.execute(query,(CustomerID,))
```

```
            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:
```

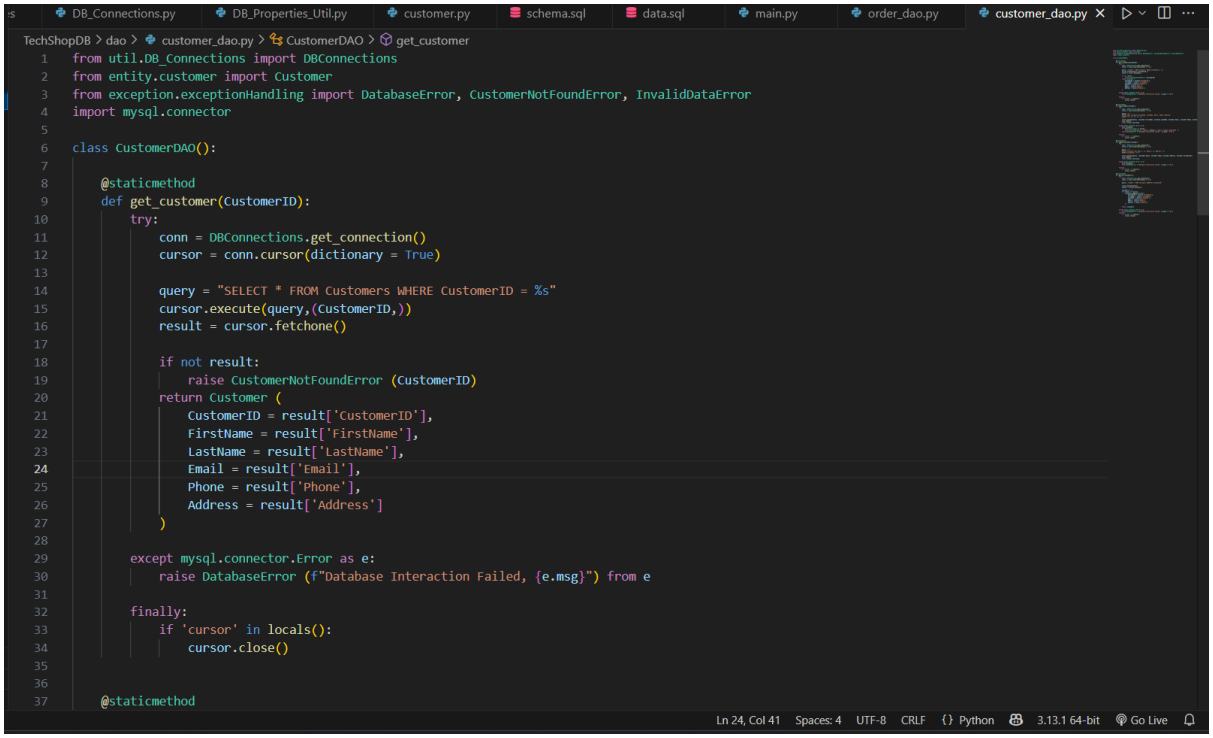
```
            inventory.append(Inventory(
                ProductName = result['ProductName'],
                ProductID = result['ProductID'],
                QuantityInStock = result['QuantityInStock']
            ))

        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()
```

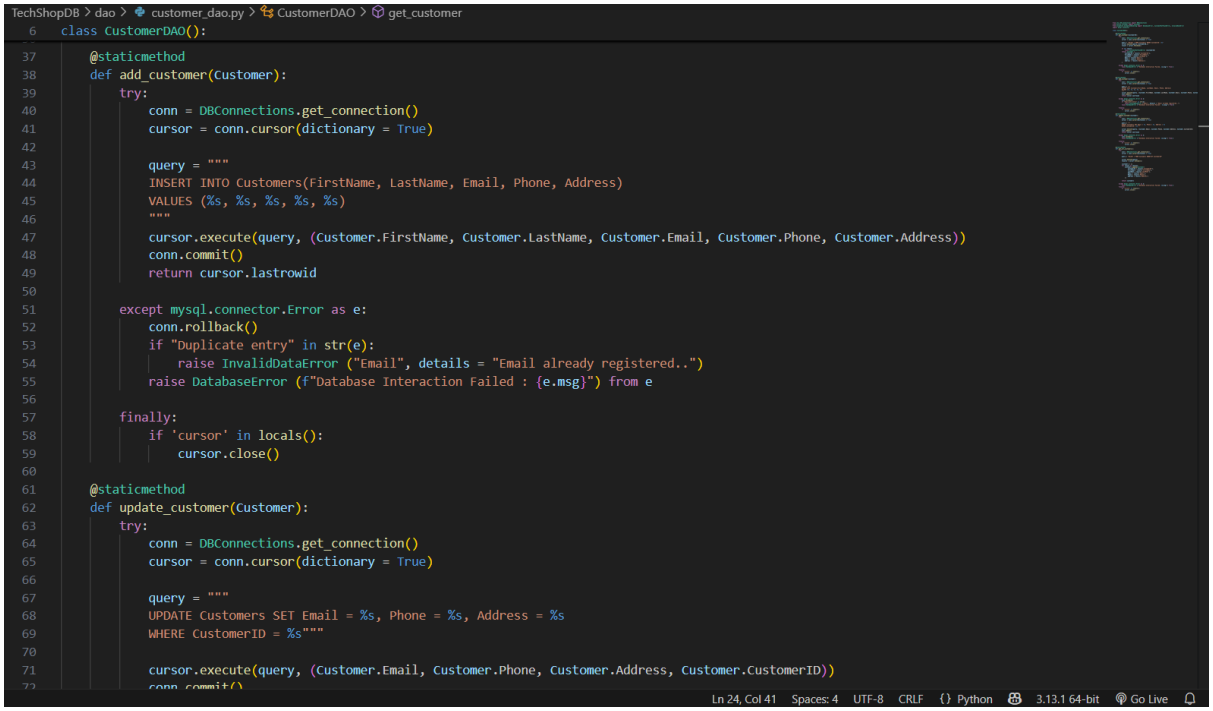
```
            raise DatabaseError(f"Failed to update inventory: {e.msg}") from e
        finally:
            if 'cursor' in locals():
                cursor.close()

    @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}")
from e
        finally:
            if 'cursor' in locals():
                cursor.close()
```

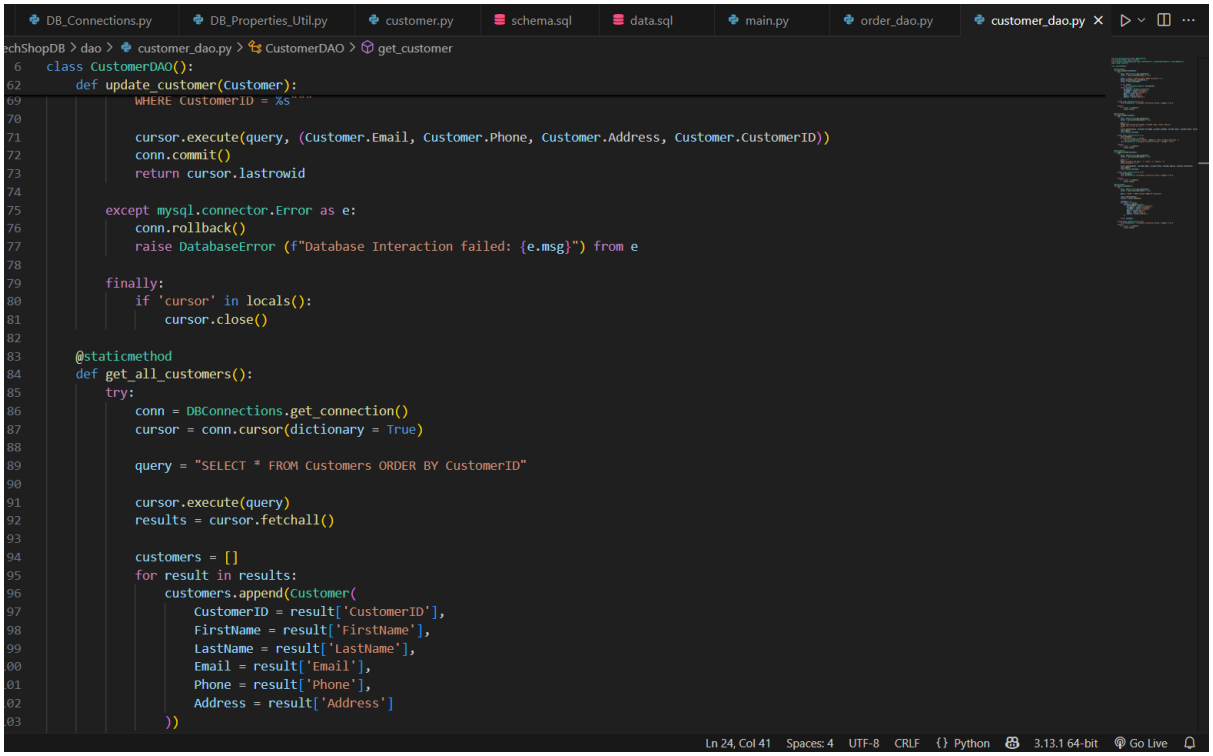
Output:

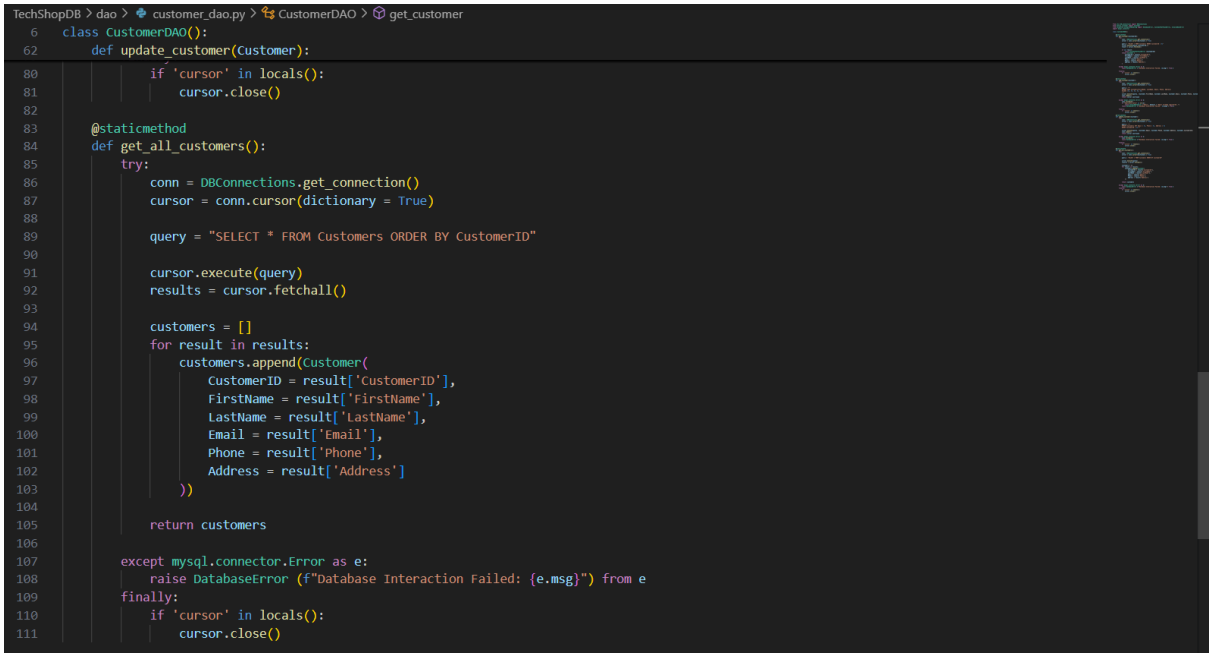


```
TechShopDB > dao > customer_dao.py > CustomerDAO > get_customer
from util.DB_Connections import DBConnections
from entity.customer import Customer
from exception.exceptionHandling import DatabaseError, CustomerNotFoundError, InvalidDataError
import mysql.connector

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
```

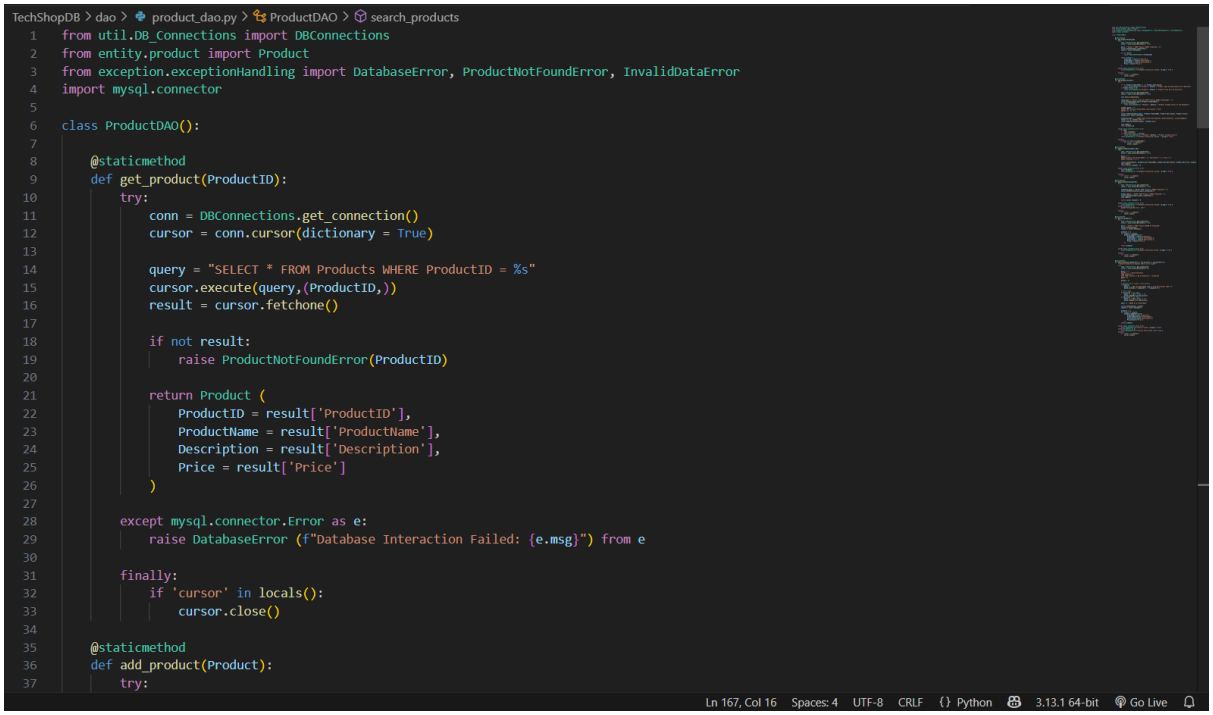


```
TechShopDB > dao > customer_dao.py > CustomerDAO > get_customer
class CustomerDAO():
    @staticmethod
    def add_customer(Customer):
        try:
            conn = DBConnections.get_connection()
            cursor = conn.cursor(dictionary = True)

            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))
```

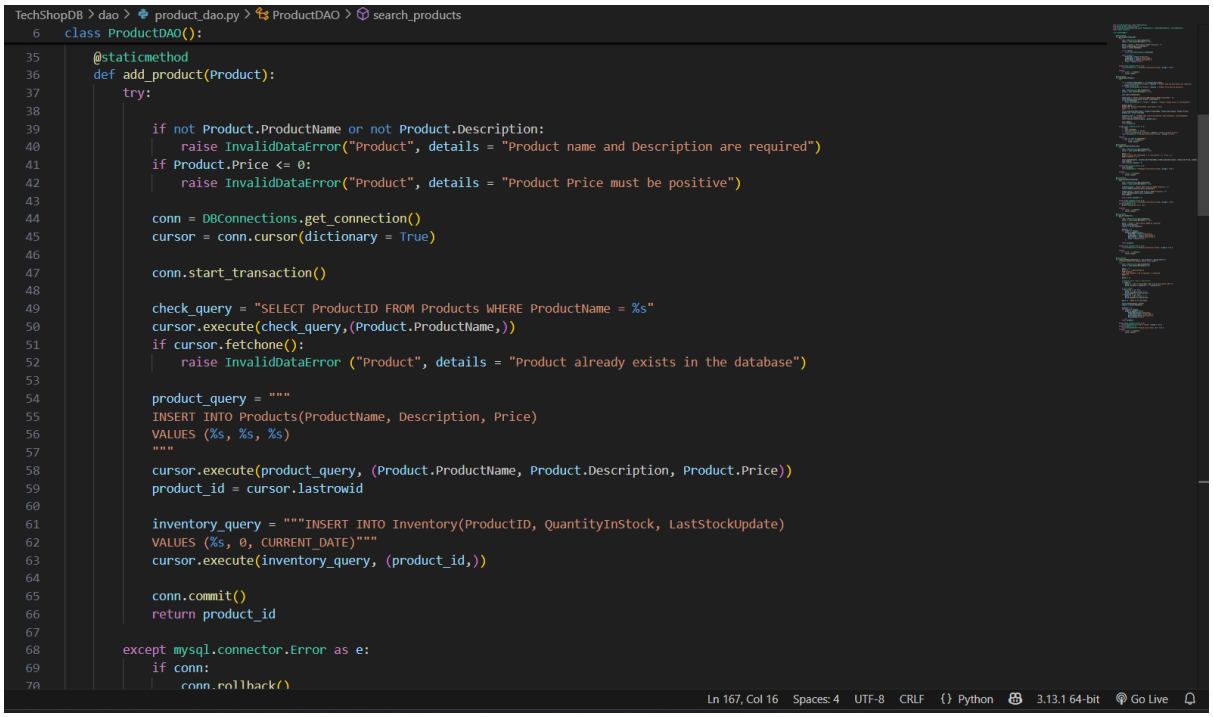




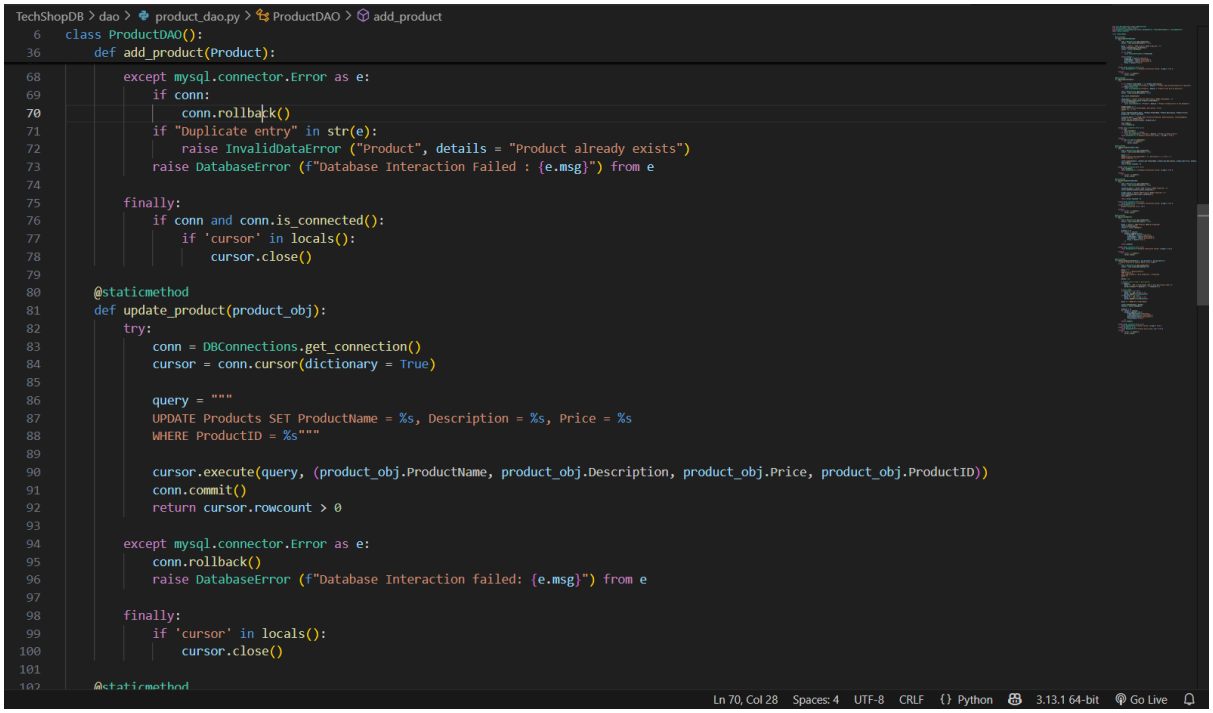
Product_dao.py:



```
TechShopDB > dao > product_dao.py > ProductDAO > search_products
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:
```

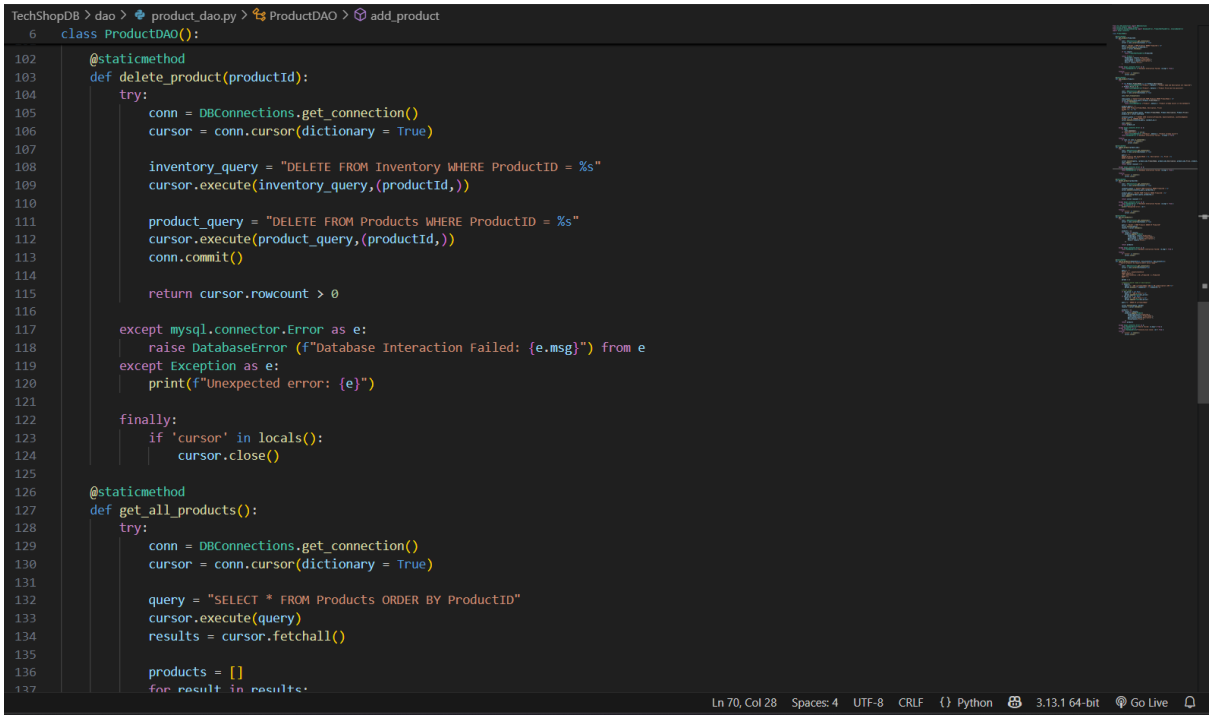



```
TechShopDB > dao > product_dao.py > ProductDAO > search_products
class ProductDAO():
    @staticmethod
    def add_product(Product):
        try:

            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()
```



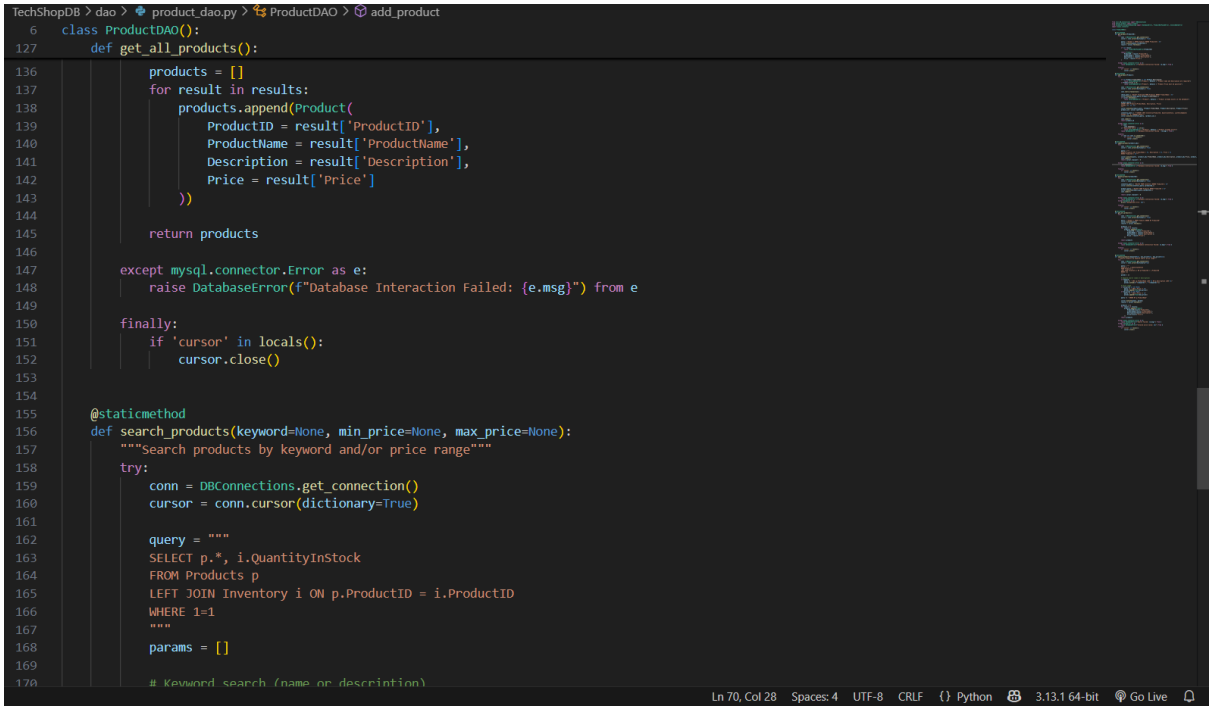
```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
  6  class ProductDAO():
 36      def add_product(Product):
 68          except mysql.connector.Error as e:
 69              if conn:
 70                  conn.rollback()
 71              if "Duplicate entry" in str(e):
 72                  raise InvalidDataError ("Product", details = "Product already exists")
 73              raise DatabaseError (f"Database Interaction Failed : {e.msg}") from e
 74
 75          finally:
 76              if conn and conn.is_connected():
 77                  if 'cursor' in locals():
 78                      cursor.close()
 79
 80      @staticmethod
 81      def update_product(product_obj):
 82          try:
 83              conn = DBConnections.get_connection()
 84              cursor = conn.cursor(dictionary = True)
 85
 86              query = """
 87              UPDATE Products SET ProductName = %s, Description = %s, Price = %s
 88              WHERE ProductID = %s"""
 89
 90              cursor.execute(query, (product_obj.ProductName, product_obj.Description, product_obj.Price, product_obj.ProductID))
 91              conn.commit()
 92              return cursor.rowcount > 0
 93
 94          except mysql.connector.Error as e:
 95              conn.rollback()
 96              raise DatabaseError (f"Database Interaction failed: {e.msg}") from e
 97
 98          finally:
 99              if 'cursor' in locals():
100                  cursor.close()
101
102      @staticmethod
```

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

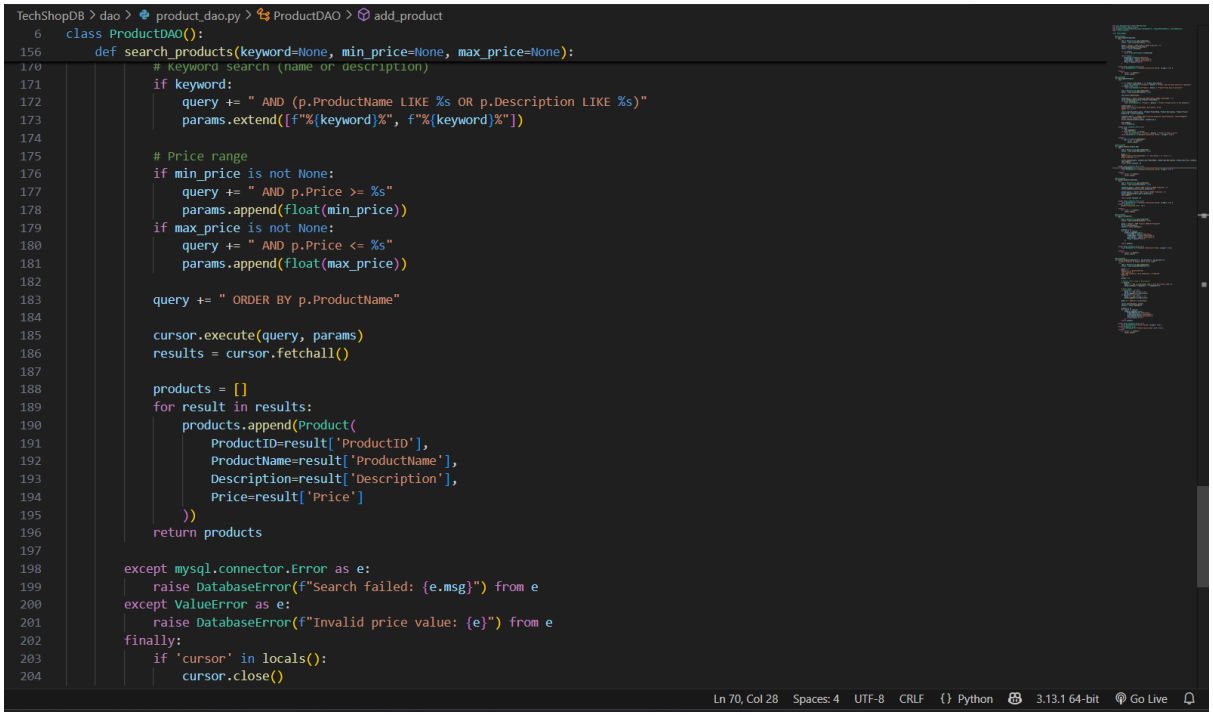


```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
  6  class ProductDAO():
102      @staticmethod
103      def delete_product(productId):
104          try:
105              conn = DBConnections.get_connection()
106              cursor = conn.cursor(dictionary = True)
107
108              inventory_query = "DELETE FROM Inventory WHERE ProductID = %s"
109              cursor.execute(inventory_query,(productId,))
110
111              product_query = "DELETE FROM Products WHERE ProductID = %s"
112              cursor.execute(product_query,(productId,))
113              conn.commit()
114
115              return cursor.rowcount > 0
116
117          except mysql.connector.Error as e:
118              raise DatabaseError (f"Database Interaction Failed: {e.msg}") from e
119          except Exception as e:
120              print(f"Unexpected error: {e}")
121
122          finally:
123              if 'cursor' in locals():
124                  cursor.close()
125
126      @staticmethod
127      def get_all_products():
128          try:
129              conn = DBConnections.get_connection()
130              cursor = conn.cursor(dictionary = True)
131
132              query = "SELECT * FROM Products ORDER BY ProductID"
133              cursor.execute(query)
134              results = cursor.fetchall()
135
136              products = []
137              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              params = []
169
170              # Keyword search (name or description)
```



```
TechShopDB > dao > product_dao.py > ProductDAO > add_product
  6  class ProductDAO():
156      def search_products(keyword=None, min_price=None, max_price=None):
170              # Keyword search (name or description)
171              if keyword:
172                  query += " AND (p.ProductName LIKE %s OR p.Description LIKE %s)"
173                  params.extend([f"%{keyword}%", f"%{keyword}%"])
174
175              # Price range
176              if min_price is not None:
177                  query += " AND p.Price >= %s"
178                  params.append(float(min_price))
179              if max_price is not None:
180                  query += " AND p.Price <= %s"
181                  params.append(float(max_price))
182
183              query += " ORDER BY p.ProductName"
184
185              cursor.execute(query, params)
186              results = cursor.fetchall()
187
188              products = []
189              for result in results:
190                  products.append(Product(
191                      ProductID=result['ProductID'],
192                      ProductName=result['ProductName'],
193                      Description=result['Description'],
194                      Price=result['Price']
195                  ))
196              return products
197
198          except mysql.connector.Error as e:
199              raise DatabaseError(f"Search failed: {e.msg}") from e
200          except ValueError as e:
201              raise DatabaseError(f"Invalid price value: {e}") from e
202          finally:
203              if 'cursor' in locals():
204                  cursor.close()
```

Order_dao.py:

TechShopDB > dao > order_dao.py > OrderDAO > get_order_count

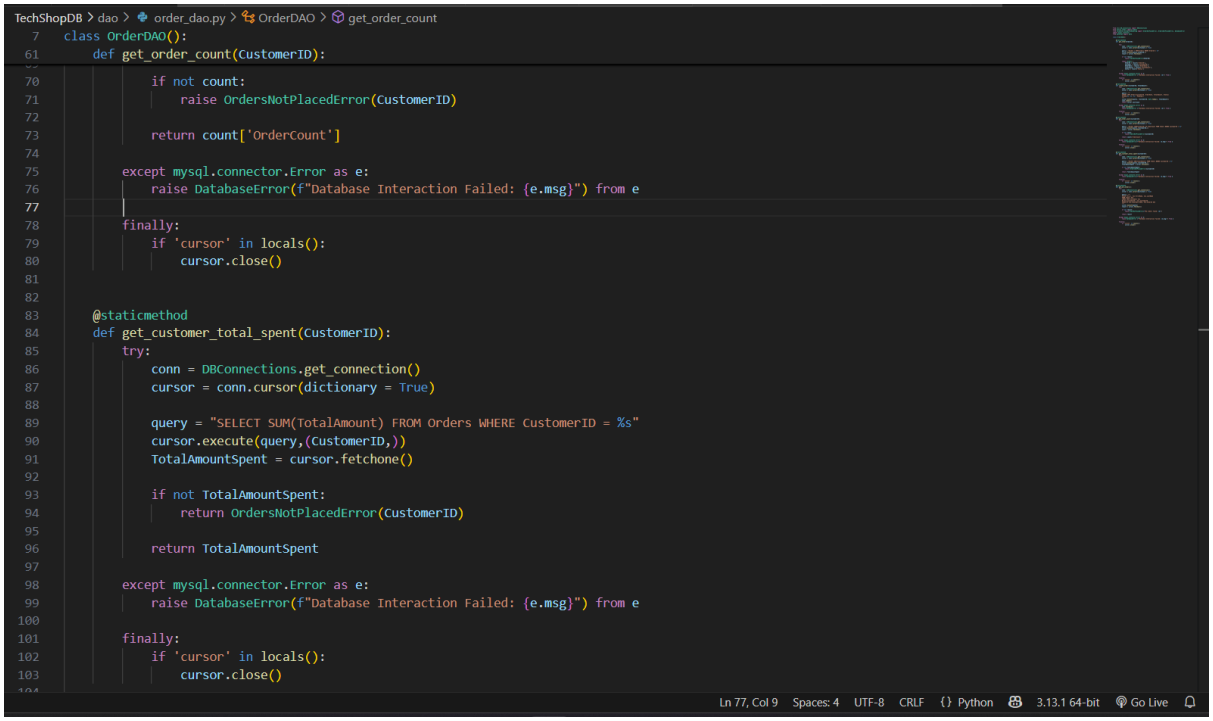
```python
from util.DB_Connections import DBConnections
from entity.orders import Orders
from exception.exceptionHandling import OrderNotFoundError, OrdersNotPlacedError, DatabaseError
import mysql.connector
from datetime import date

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()

```

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

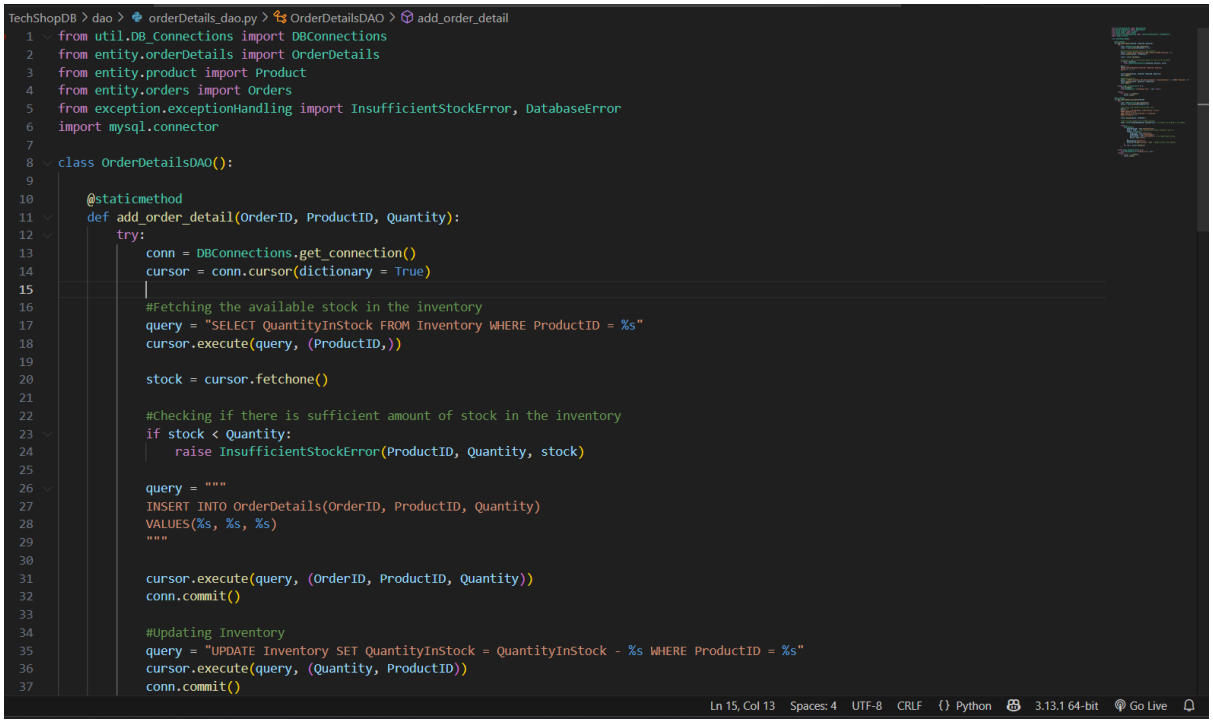
TechShopDB > dao > order_dao.py > OrderDAO > get_order_count

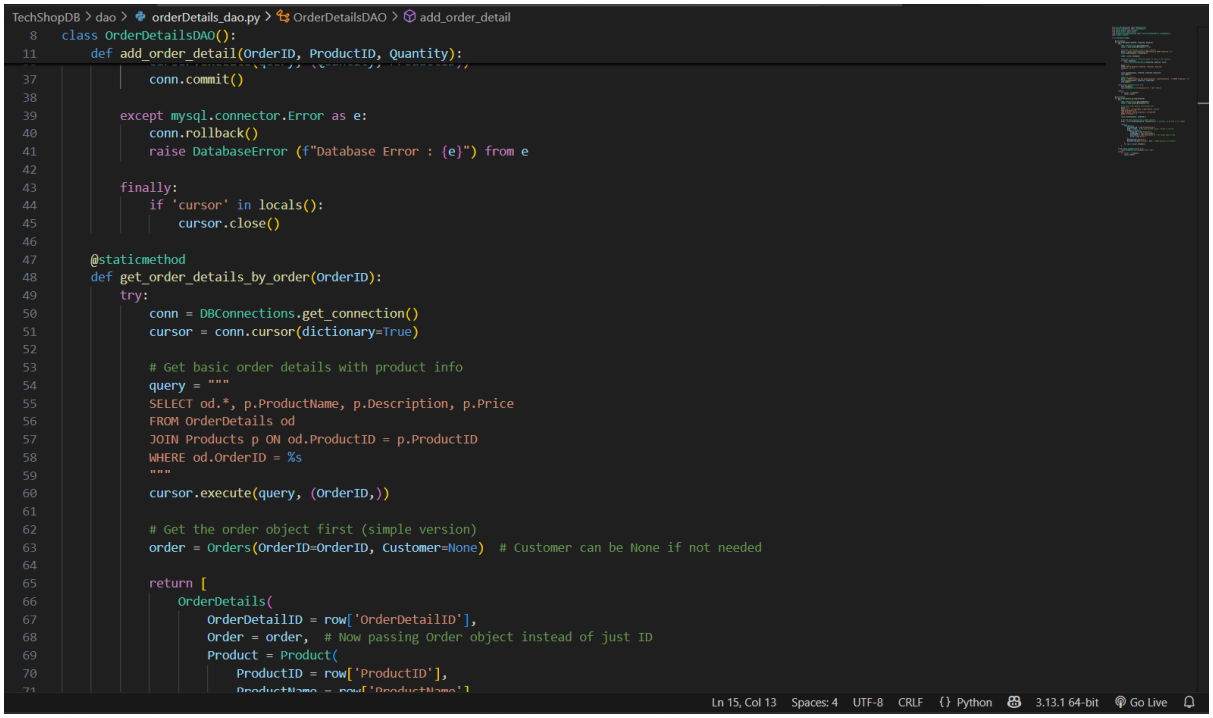
```python
class OrderDAO():

    @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)

            query = "SELECT COUNT(OrderID) AS OrderCount FROM Orders WHERE CustomerID = %s"
            cursor.execute(query,(CustomerID,))
            count = cursor.fetchone()

            if not count:
                raise OrdersNotPlacedError(CustomerID)

```

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



orderDetails_dao.py:






```
                    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:

TechShopDB > dao > inventory_dao.py > InventoryDAO > get_inventory

```python
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']
            )
```

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

```python
class InventoryDAO():
    def get_inventory(ProductID):
        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()
```

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

```python
class InventoryDAO():
    @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
```

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

```python
class InventoryDAO():
    def update_inventory(inventory):
            ))
            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():
```

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

```python
    @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}") from e
        finally:
            if 'cursor' in locals():
                cursor.close()
```

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:
```

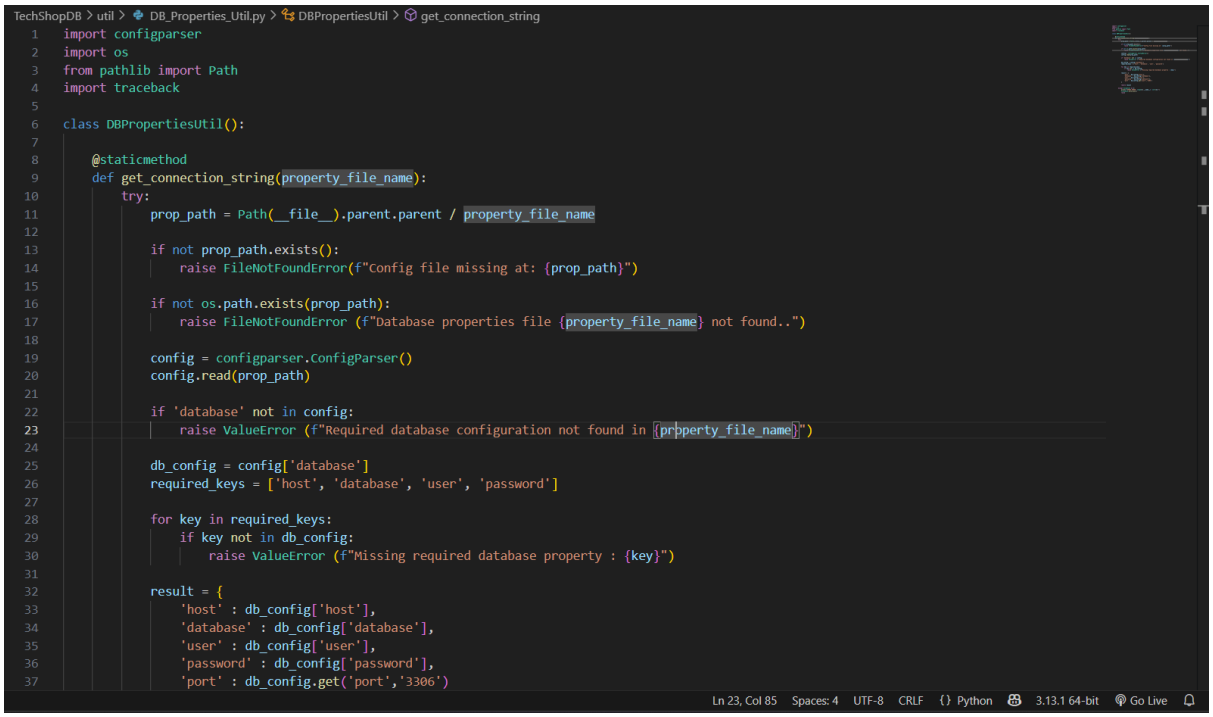
```
            print(f"DEBUG ERROR: {type(e).__name__}: {str(e)}")
            traceback.print_exc()
            raise
```

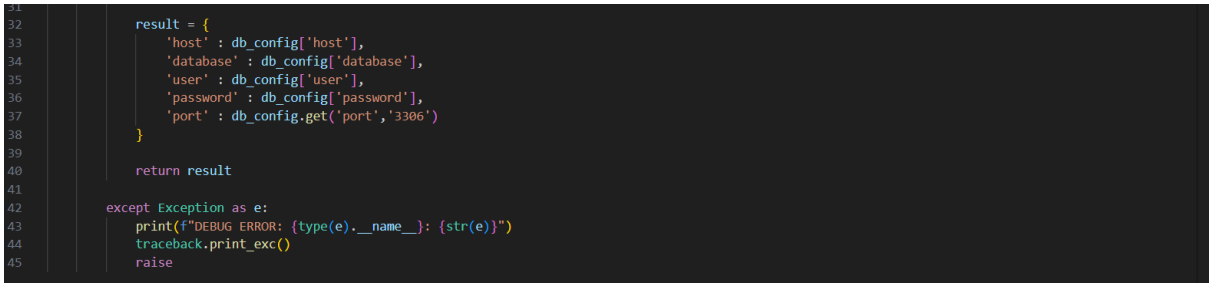
DB_Connections.py:

```
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_Connections.py > DBConnections > get_connection

```python
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

```

1: Customer Registration:

Query:

```python
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
```

```python
@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()
```

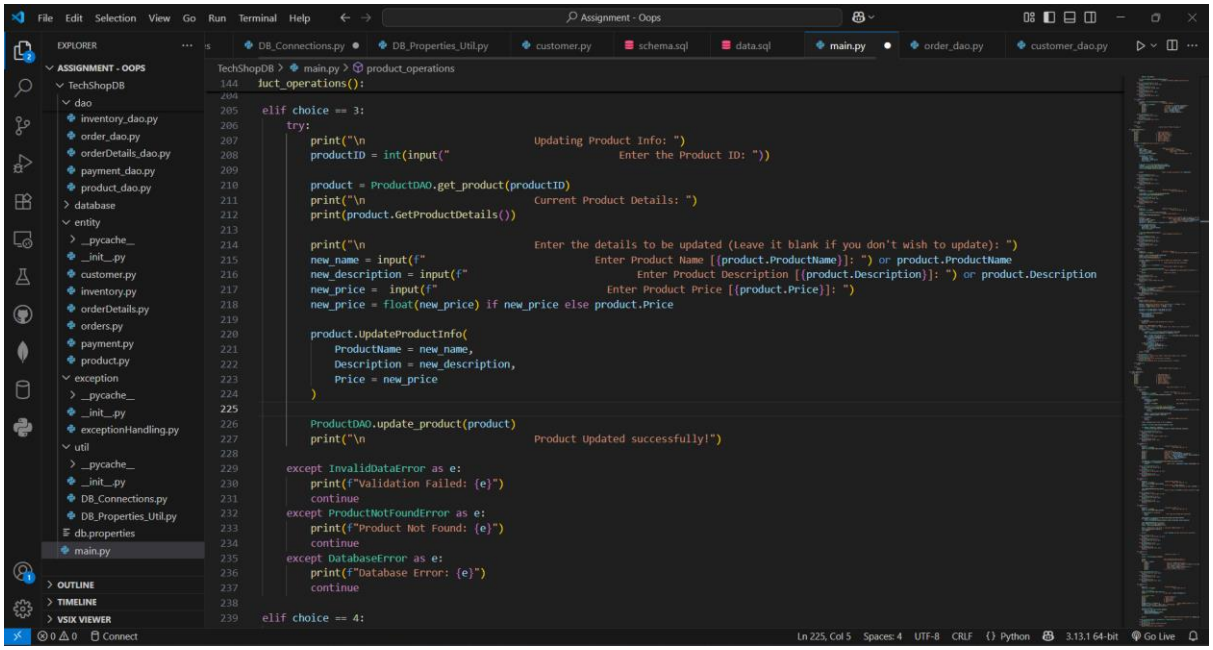
Output:

```
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: caren24@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
                    LastName: Smith
                    Email: caren24@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):
```

2. Product Catalog Management:

Query:



```
elif choice == 3:
    try:
        print("\n                    Updating Product Info: ")
        productID = int(input("                        Enter the Product ID: "))

        product = ProductDAO.get_product(productID)
        print("\n                    Current Product Details: ")
        print(product.GetProductDetails())

        print("\n                    Enter the details to be updated (Leave it blank if you don't wish to update): ")
        new_name = input(f"                Enter Product Name [{product.ProductName}]: ") or product.ProductName
        new_description = input(f"            Enter Product Description [{product.Description}]: ") or product.Description
        new_price =  input(f"              Enter Product Price [{product.Price}]: ")
        new_price = float(new_price) if new_price else product.Price

        product.UpdateProductInfo(
            ProductName = new_name,
            Description = new_description,
            Price = new_price
        )

        ProductDAO.update_product(product)
        print("\n                    Product Updated successfully!")

    except InvalidDataError as e:
        print(f"Validation Failed: {e}")
        continue
    except ProductNotFoundError as e:
        print(f"Product Not Found: {e}")
        continue
    except DatabaseError as e:
        print(f"Database Error: {e}")
        continue

elif choice == 4:
```

```
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
```

```python
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()
```

```python
@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()
```


Output:

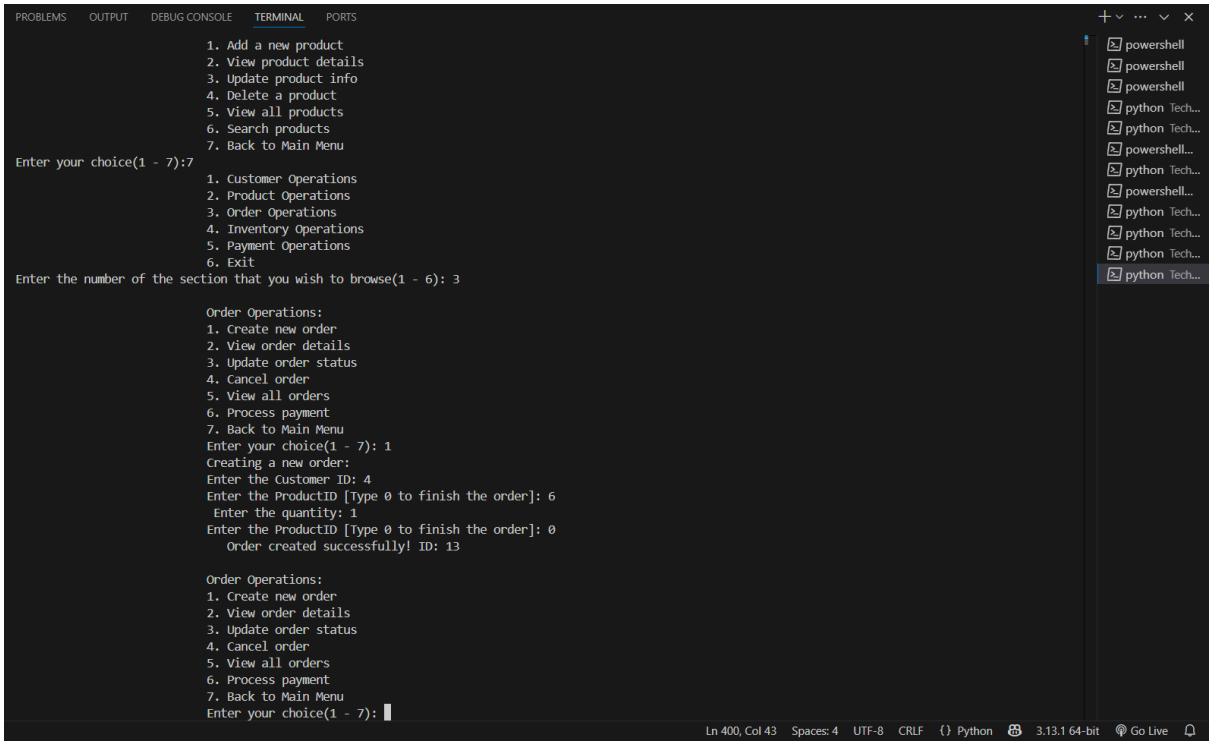
```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Assignment\Assignment - Oops\TechShopDB> python main.py
                        Welcome to TechShop Management System
                        1. Customer Operations
                        2. Product Operations
                        3. Order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

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

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

```python
@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()
        ]
```

```python
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                          Adding a new product: ")
        name = input("                          Product Name: ").strip()
        description = input("                          Description: ").strip()
        price = float(input("                          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(f"                             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:\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):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):
```

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

```python
555    def inventory_operations():
718            elif choice == 7:
719                try:
720                    print("\nGenerating Sales Summary Report...")
721
722                    conn = None
723                    cursor = None
724                    try:
725                        conn = DBConnections.get_connection()
726                        cursor = conn.cursor(dictionary=True)
727
728                        # Total sales query
729                        cursor.execute("""
730                        SELECT
731                            COUNT(DISTINCT OrderID) AS total_orders,
732                            SUM(TotalAmount) AS total_revenue,
733                            AVG(TotalAmount) AS avg_order_value
734                        FROM Orders
735                        """)
736                        sales_data = cursor.fetchone()
737
738                        # Top products query
739                        cursor.execute("""
740                        SELECT p.ProductName, SUM(od.Quantity) AS units_sold
741                        FROM OrderDetails od
742                        JOIN Products p ON od.ProductID = p.ProductID
743                        GROUP BY p.ProductID
744                        ORDER BY units_sold DESC
745                        LIMIT 3
746                        """)
747                        top_products = cursor.fetchall()
748
749                        # Display report
750                        print("\n=== SALES SUMMARY ===")
751                        print(f"Total Orders: {sales_data['total_orders']}")
752                        print(f"Total Revenue: Rs.{sales_data['total_revenue']:,.2f}")
753                        print(f"Avg Order Value: Rs.{sales_data['avg_order_value']:,.2f}")
```

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

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

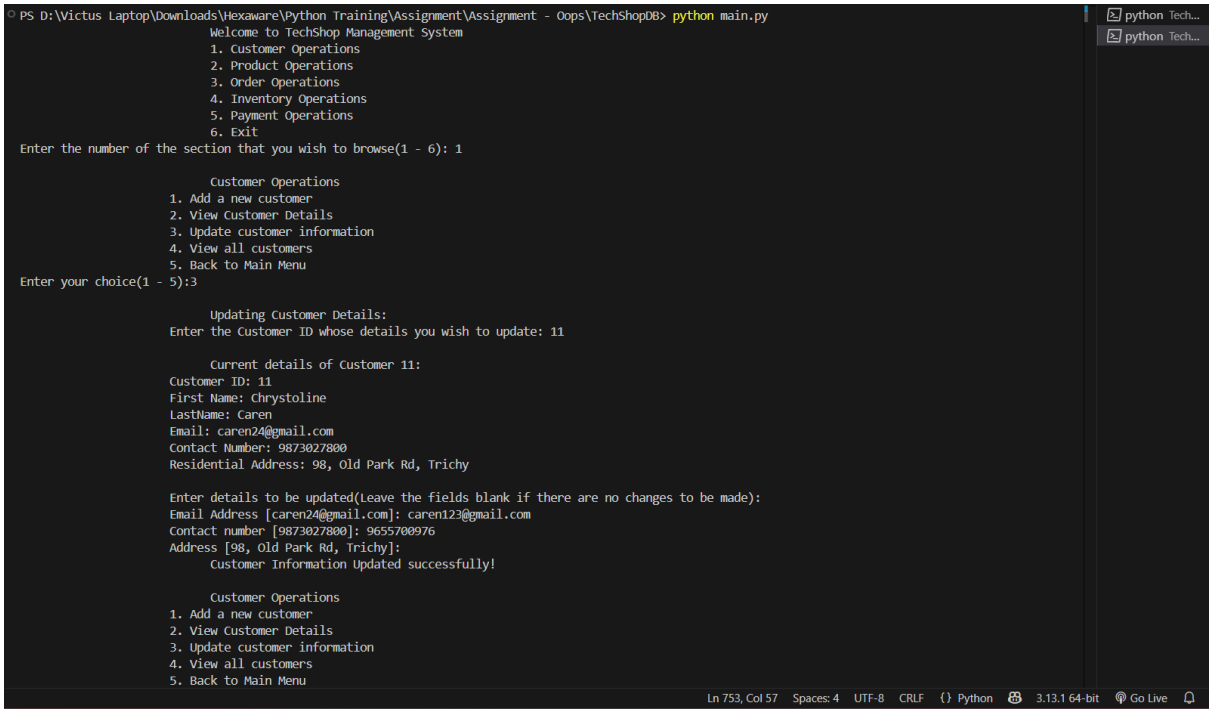
Query:

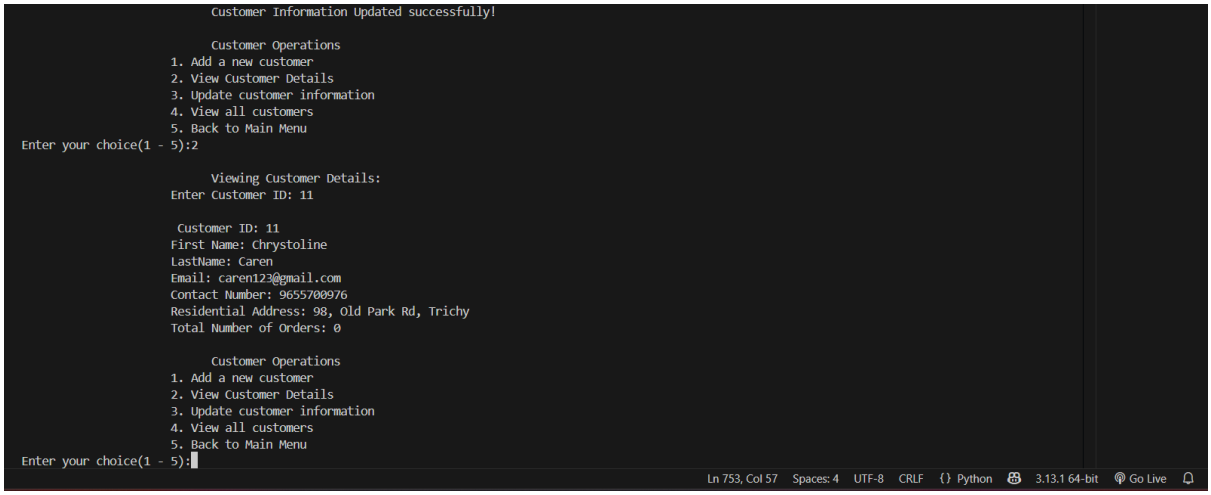
```
if customer_operations():

    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 = %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()
```

Output:





8: Payment Processing

Query:

```python
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("\n                          All 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:
```

TechShopDB > entity > payment.py > Payment > GetPaymentDetails

```python
class Payment:
    @property
    def Amount(self):
        return self.__Amount

    @property
    def Status(self):
        return self.__Status

    # Setters with validation
    @PaymentMethod.setter
    def PaymentMethod(self, value):
        valid_methods = ['Credit Card', 'Debit Card', 'Net Banking', 'UPI', 'Cash']
        if value not in valid_methods:
            raise InvalidDataError("Payment Method", f"Must be one of: {', '.join(valid_methods)}")
        self.__PaymentMethod = value

    @Amount.setter
    def Amount(self, value):
        if value <= 0:
            raise InvalidDataError("Amount", "Must be greater than 0")
        self.__Amount = value

    @Status.setter
    def Status(self, value):
        valid_statuses = ['Pending', 'Completed', 'Failed', 'Refunded']
        if value not in valid_statuses:
            raise InvalidDataError("Status", f"Must be one of: {', '.join(valid_statuses)}")
        self.__Status = value

    def GetPaymentDetails(self):
        return f"                          Payment ID: {self.__PaymentID}\n                                                  Order ID: {self.__OrderID}\n
```

```python
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:
```

```python
            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} "
```

```python
def product_operations():
                )

                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 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 - 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):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
```