TECHSHOP-ELECTRONIC GADGETS

K. MEGHASYAM

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:

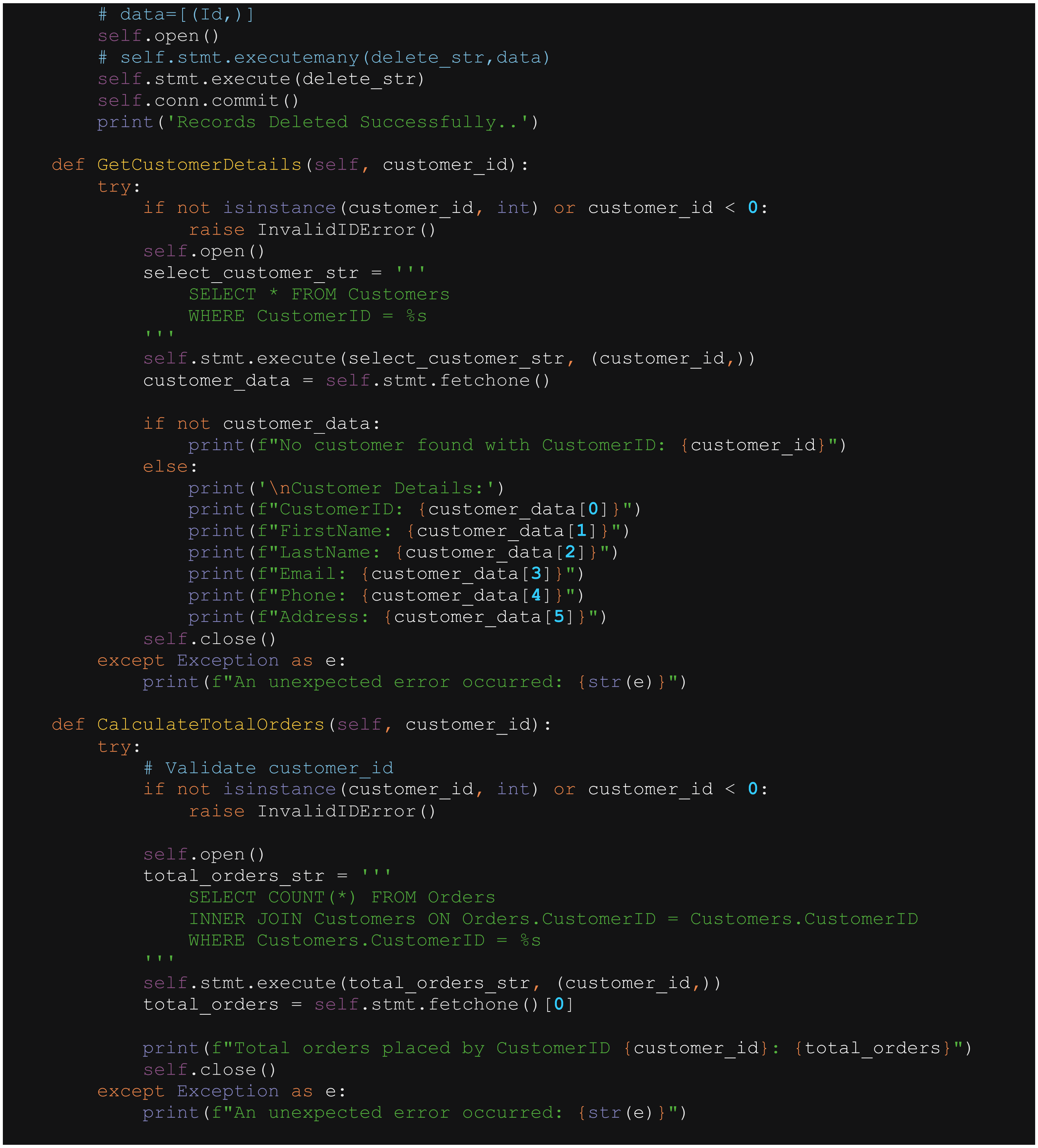
- CustomerID (int)
- FirstName (string)
- LastName (string)
- Email (string)
- Phone (string)
- Address (string)

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,)

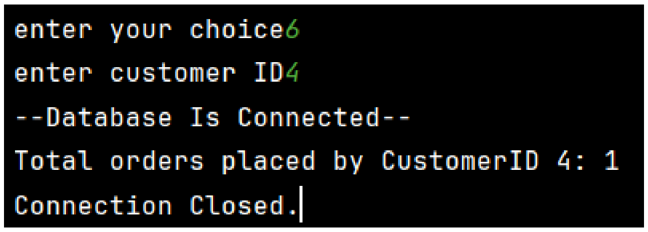
```
from exceptions.InvalidDataException import InvalidEmailError , validate_email
from exceptions.InvalidIDError import InvalidIDError
from exceptions.InvalidNameError import InvalidNameError , StringCheck
from exceptions.InvalidPhoneError import InvalidPhoneError , validate_phone
from util.DBConnUtil import dbConnection
class Customers(dbConnection):

    def __init__(self):
        self.name = ''
        self.email = ''
        self.phone = ''
        print(self.name, self.email, self.name)

    def create(self):
        create_str = '''
        CREATE TABLE IF NOT EXISTS Customers (
        CustomerID INT PRIMARY KEY AUTO_INCREMENT,
        FirstName VARCHAR(55),
        LastName VARCHAR(55),
        Email VARCHAR(55),
        Phone VARCHAR(20),
        Address VARCHAR(55)
                );
        '''
        self.open()
        self.stmt.execute(create_str)
        self.stmt.close()
        print('Customers Table created successfully------:')

    def addCustomer(self):

        first_name = input('Enter First Name :')
        if not isinstance(first_name, str):
            raise InvalidNameError()
        StringCheck(first_name)
        self.firstname = first_name

        last_name = input('Enter Last Name :')
        if not isinstance(last_name, str):
            raise InvalidNameError()
        StringCheck(last_name)
        self.lastname = last_name

        Email = input('Enter email:')
        if not isinstance(Email, str):
            raise InvalidEmailError()
        validate_email(Email)
        self.email = Email

        Phone = input('Enter phone :')
        if not isinstance(Phone, str):
            raise InvalidPhoneError()
        validate_phone(Phone)
        self.phone = Phone

        Address = input('Enter address :')
        self.address = Address

        numberoforders = input('Enter orders :')
        self.numberoforders = numberoforders

        data = [(self.firstname, self.lastname, self.email, self.phone, self.address,
self.numberoforders)]
        insert_str = '''insert into
Customers(FirstName,LastName,Email,Phone,Address,numberoforders)
        values(%s,%s,%s,%s,%s,%s)'''
        self.open()
        self.stmt.executemany(insert_str, data)
        self.conn.commit()
        print('Records Inserted Successfully..')
        self.close()

    def select(self):
        self.open()
        select_str = '''select * from customers'''
        self.stmt.execute(select_str)
        recods = self.stmt.fetchall()
        print('')
        print('_________________Records In Customer Table________________________')
        for i in recods:
            print(i)
        self.close()
        return f'Customers details fetched successfully'


    def UpdateCustomerInfo(self):
        self.select()
```

```python
        customer_id = int(input('Input Customer ID to be Updated: '))
        if not isinstance(customer_id, int) :
            raise InvalidIDError()
        else:
            if customer_id < 0:
                raise InvalidIDError()
        Id = customer_id
        update_str = 'UPDATE customers SET '
        data = []

        first_name = input('Enter First Name ((Press Enter to skip)):')
        if first_name:
            if not isinstance(first_name, str):
                raise InvalidNameError()
            StringCheck(first_name)
            self.firstname = first_name
            update_str += 'FirstName=%s, '
            data.append(self.firstname)

        last_name = input('Enter Last Name ((Press Enter to skip)):')
        if last_name:
            if not isinstance(last_name, str):
                raise InvalidNameError()
            StringCheck(last_name)
            self.lastname = last_name
            update_str += 'LastName=%s, '
            data.append(self.lastname)

        Email = input('Enter email:(Press Enter to skip)')
        if Email:
            if not isinstance(Email, str):
                raise InvalidEmailError()
            validate_email(Email)
            self.email = Email
            update_str += 'Email=%s, '
            data.append(self.email)

        Phone = input('Enter Phone (Press Enter to skip): ')
        if Phone:
            if not isinstance(Phone, str):
                raise InvalidPhoneError()
            validate_phone(Phone)
            self.phone = Phone
            update_str += 'Phone=%s, '
            data.append(self.phone)

        update_str = update_str.rstrip(', ')

        update_str += ' WHERE CustomerID=%s'
        data.append(Id)

        self.open()
        self.stmt.execute(update_str, data)
        self.conn.commit()
        print('Record updated successfully.')
        self.select()

    def delete(self):
        customer_id = int(input('Input Customer ID to be Updated: '))
        if not isinstance(customer_id, int) or customer_id < 0:
            raise InvalidIDError()
        Id = customer_id
        delete_str = f'delete from customers where CustomerID={Id}'
```



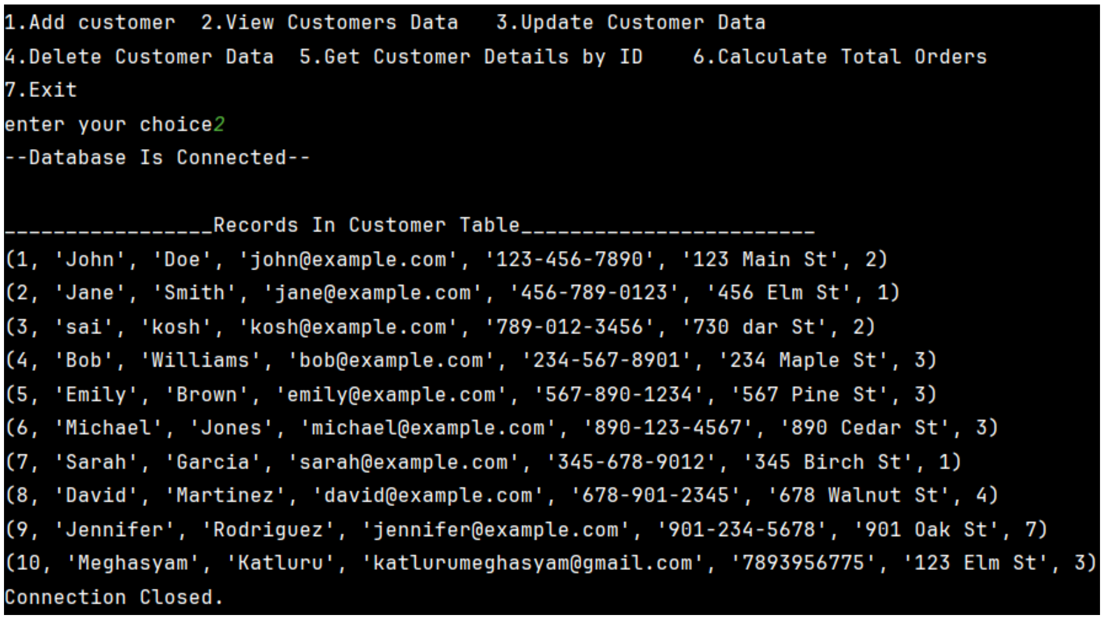
```python
        # data=[(Id,)]
        self.open()
        # self.stmt.executemany(delete_str,data)
        self.stmt.execute(delete_str)
        self.conn.commit()
        print('Records Deleted Successfully..')

    def GetCustomerDetails(self, customer_id):
        try:
            if not isinstance(customer_id, int) or customer_id < 0:
                raise InvalidIDError()
            self.open()
            select_customer_str = '''
                SELECT * FROM Customers
                WHERE CustomerID = %s
            '''
            self.stmt.execute(select_customer_str, (customer_id,))
            customer_data = self.stmt.fetchone()

            if not customer_data:
                print(f"No customer found with CustomerID: {customer_id}")
            else:
                print('\nCustomer Details:')
                print(f"CustomerID: {customer_data[0]}")
                print(f"FirstName: {customer_data[1]}")
                print(f"LastName: {customer_data[2]}")
                print(f"Email: {customer_data[3]}")
                print(f"Phone: {customer_data[4]}")
                print(f"Address: {customer_data[5]}")
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def CalculateTotalOrders(self, customer_id):
        try:
            # Validate customer_id
            if not isinstance(customer_id, int) or customer_id < 0:
                raise InvalidIDError()

            self.open()
            total_orders_str = '''
                SELECT COUNT(*) FROM Orders
                INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID
                WHERE Customers.CustomerID = %s
            '''
            self.stmt.execute(total_orders_str, (customer_id,))
            total_orders = self.stmt.fetchone()[0]

            print(f"Total orders placed by CustomerID {customer_id}: {total_orders}")
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")
```

CalculateTotalOrders O/P:



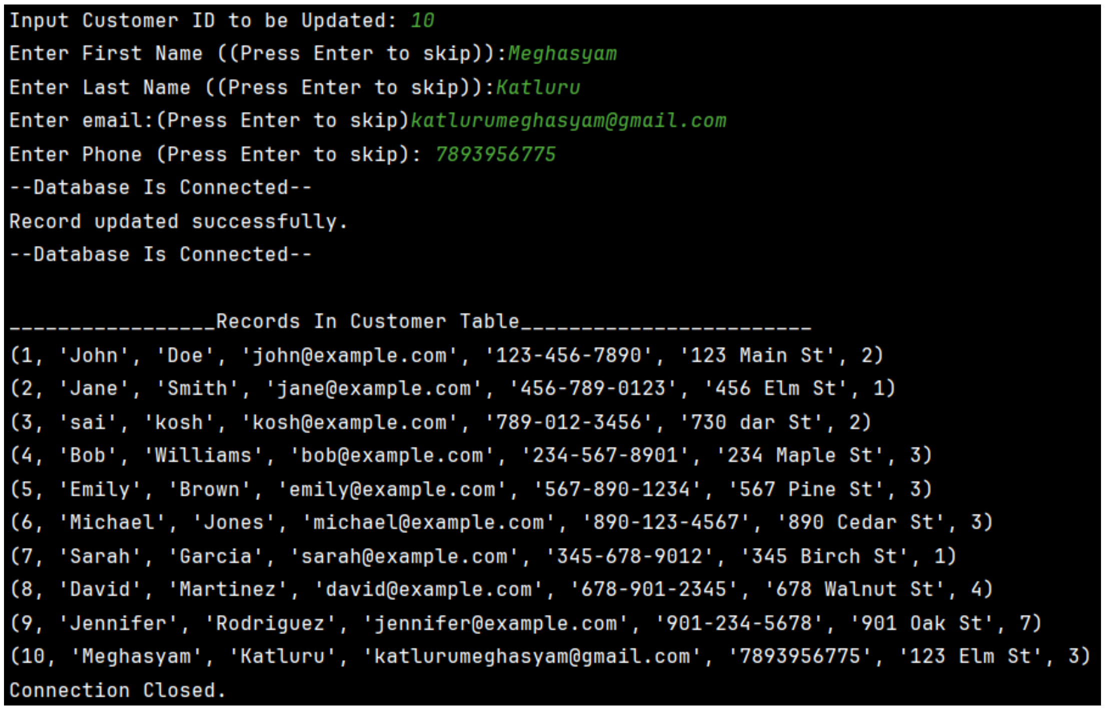
```
enter your choice6
enter customer ID4
--Database Is Connected--
Total orders placed by CustomerID 4: 1
Connection Closed.
```

GetCustomerDetails O/P:



```
1.Add customer  2.View Customers Data   3.Update Customer Data
4.Delete Customer Data  5.Get Customer Details by ID    6.Calculate Total Orders
7.Exit
enter your choice2
--Database Is Connected--

_________________Records In Customer Table________________________
(1, 'John', 'Doe', 'john@example.com', '123-456-7890', '123 Main St', 2)
(2, 'Jane', 'Smith', 'jane@example.com', '456-789-0123', '456 Elm St', 1)
(3, 'sai', 'kosh', 'kosh@example.com', '789-012-3456', '730 dar St', 2)
(4, 'Bob', 'Williams', 'bob@example.com', '234-567-8901', '234 Maple St', 3)
(5, 'Emily', 'Brown', 'emily@example.com', '567-890-1234', '567 Pine St', 3)
(6, 'Michael', 'Jones', 'michael@example.com', '890-123-4567', '890 Cedar St', 3)
(7, 'Sarah', 'Garcia', 'sarah@example.com', '345-678-9012', '345 Birch St', 1)
(8, 'David', 'Martinez', 'david@example.com', '678-901-2345', '678 Walnut St', 4)
(9, 'Jennifer', 'Rodriguez', 'jennifer@example.com', '901-234-5678', '901 Oak St', 7)
(10, 'Meghasyam', 'Katluru', 'katlurumeghasyam@gmail.com', '7893956775', '123 Elm St', 3)
Connection Closed.
```

UpdateCustomerInfo O/P:



```
Input Customer ID to be Updated: 10
Enter First Name ((Press Enter to skip)):Meghasyam
Enter Last Name ((Press Enter to skip)):Katluru
Enter email:(Press Enter to skip)katlurumeghasyam@gmail.com
Enter Phone (Press Enter to skip): 7893956775
--Database Is Connected--
Record updated successfully.
--Database Is Connected--

_________________Records In Customer Table________________________
(1, 'John', 'Doe', 'john@example.com', '123-456-7890', '123 Main St', 2)
(2, 'Jane', 'Smith', 'jane@example.com', '456-789-0123', '456 Elm St', 1)
(3, 'sai', 'kosh', 'kosh@example.com', '789-012-3456', '730 dar St', 2)
(4, 'Bob', 'Williams', 'bob@example.com', '234-567-8901', '234 Maple St', 3)
(5, 'Emily', 'Brown', 'emily@example.com', '567-890-1234', '567 Pine St', 3)
(6, 'Michael', 'Jones', 'michael@example.com', '890-123-4567', '890 Cedar St', 3)
(7, 'Sarah', 'Garcia', 'sarah@example.com', '345-678-9012', '345 Birch St', 1)
(8, 'David', 'Martinez', 'david@example.com', '678-901-2345', '678 Walnut St', 4)
(9, 'Jennifer', 'Rodriguez', 'jennifer@example.com', '901-234-5678', '901 Oak St', 7)
(10, 'Meghasyam', 'Katluru', 'katlurumeghasyam@gmail.com', '7893956775', '123 Elm St', 3)
Connection Closed.
```

Products Class:

Attributes:

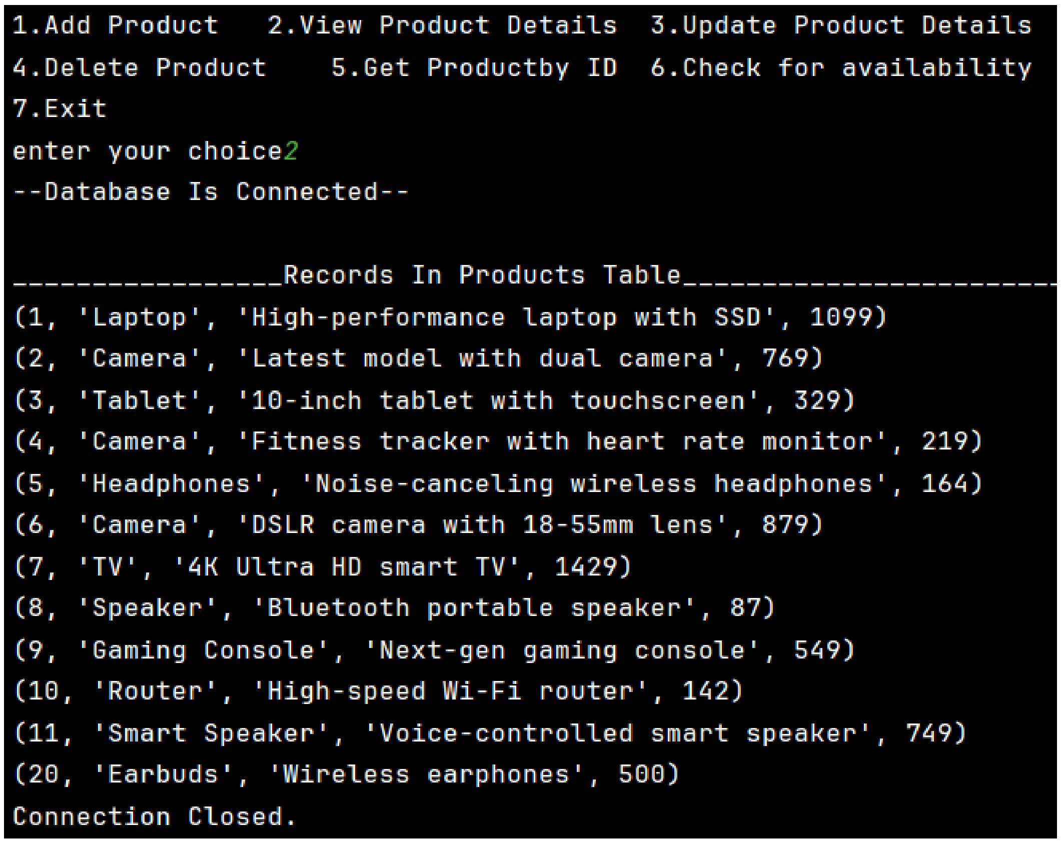
- ProductID (int)
- ProductName (string)
- Description (string)
- Price (decimal)

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.

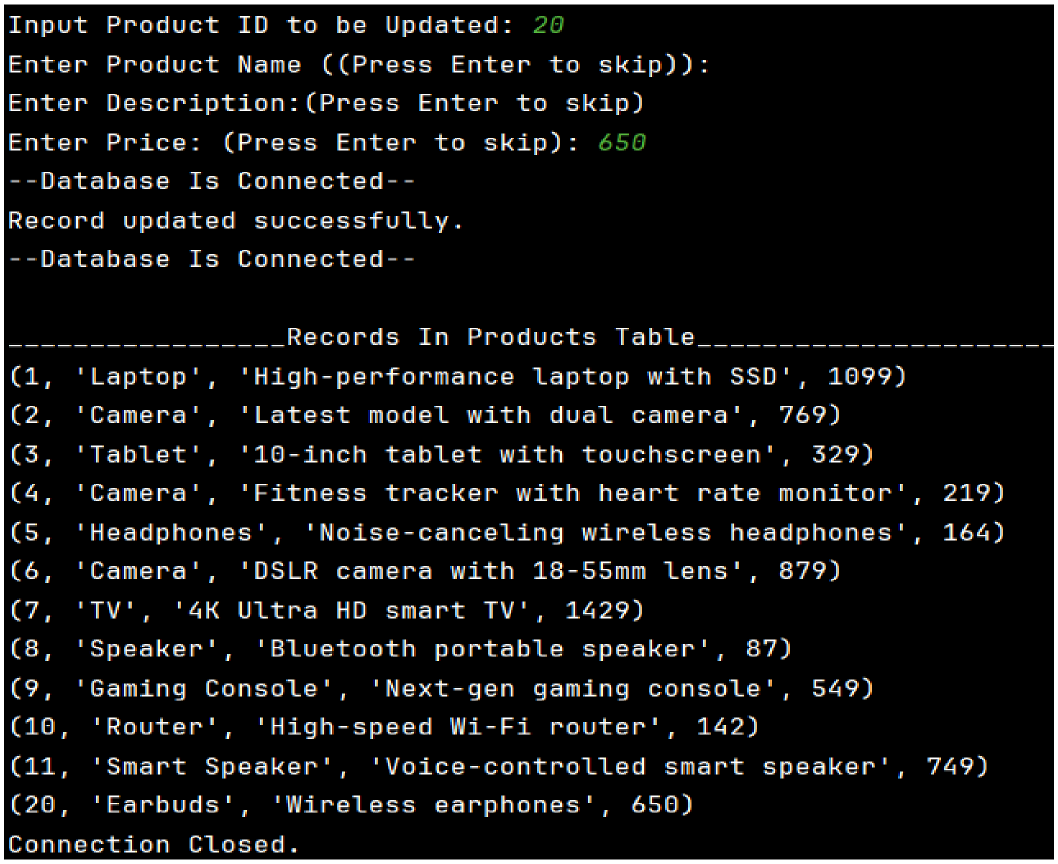
```python
from exceptions.InvalidIDError import InvalidIDError
from exceptions.InvalidNameError import InvalidNameError , StringCheck
from exceptions.InvalidPriceError import InvalidPriceError
from util.DBConnUtil import dbConnection


class Products(dbConnection):

    def create(self):
        create_str = '''
        CREATE TABLE IF NOT EXISTS Products (
            ProductID INT PRIMARY KEY AUTO_INCREMENT,
            ProductName VARCHAR(55),
            Description VARCHAR(55),
            Price INT
        );
        '''
        self.open()
        self.stmt.execute(create_str)
        self.stmt.close()
        print('Products Table created successfully------:')

    def addProduct(self):
        Product_Name = input('Enter Product Name :')
        if not isinstance(Product_Name, str):
            raise InvalidNameError()
        StringCheck(Product_Name)
        self.Product_Name = Product_Name

        Description = input('Enter Description:')
        self.Description = Description

        new = int(input('Enter Price:'))
        if not isinstance(new, int) or new<0:
            raise InvalidPriceError()
        self.price = new

        data = [(self.Product_Name, self.Description, self.price)]
        insert_str = '''INSERT INTO Products (Product_Name, Description, Price)
                        VALUES (%s, %s, %s)'''

        self.open()
        self.stmt.executemany(insert_str, data)
        self.conn.commit()
        print('Records Inserted Successfully..')
        self.close()

    def selectProducts(self):
        self.open()

        select_str = '''SELECT * FROM Products'''
        self.stmt.execute(select_str)
        records = self.stmt.fetchall()
        print('')
        print('_________________Records In Products Table________________________')
        for record in records:
            print(record)
        self.close()
        return f'Products details fetched successfully'

    def UpdateProductInfo(self):
        self.selectProducts()
        Product_id = int(input('Input Product ID to be Updated: '))
        if not isinstance(Product_id, int) or Product_id<0:
            raise InvalidIDError()
        Id = Product_id
        update_str = 'UPDATE Products SET '
        data = []

        Product_name = input('Enter Product Name ((Press Enter to skip)):')
        if Product_name:
            if not isinstance(Product_name, str):
                raise InvalidNameError()
            StringCheck(Product_name)
            self.ProductName = Product_name
            update_str += 'ProductName=%s, '
            data.append(self.ProductName)

        Description = input('Enter Description:(Press Enter to skip)')
        if Description:
            self.Description = Description
            update_str += 'Description=%s, '
            data.append(self.Description)

        Price = input('Enter Price: (Press Enter to skip): ')
        if Price:
            Price = int(Price)
            if not isinstance(Price, (int, float)) or Price < 0:
                raise InvalidPriceError()
            self.Price = int(Price)
            update_str += 'Price=%s, '
            data.append(self.Price)

        update_str = update_str.rstrip(', ')

        update_str += ' WHERE Product_ID=%s'
        data.append(Id)

        self.open()
        self.stmt.execute(update_str, data)
        self.conn.commit()
        print('Record updated successfully.')
        self.selectProducts()

    def deleteProduct(self):
        self.selectProducts()
        Product_id = int(input('Input Product ID to be Deleted: '))
        if not isinstance(Product_id, int):
            raise InvalidIDError()
        else:
            if Product_id < 0:
                raise InvalidIDError()
        Id = Product_id
        delete_str = f'DELETE FROM Products WHERE Product_ID={Id}'
        self.open()
        self.stmt.execute(delete_str)
        self.conn.commit()
        self.close()
        print('Record Deleted Successfully..')

    def GetProductDetails(self, product_id):
        try:
            # Validate product_id
            if not isinstance(product_id, int) or product_id < 0:
                raise InvalidIDError()

            self.open()
            get_product_details_str = '''
                SELECT * FROM Products
                WHERE Product_ID = %s
            '''
            self.stmt.execute(get_product_details_str, (product_id,))
            product_data = self.stmt.fetchone()

            if not product_data:
                print(f"No product found with ProductID: {product_id}")
            else:
                print('\nProduct Details:')
                print(f"ProductID: {product_data[0]}")
                print(f"ProductName: {product_data[1]}")
                print(f"Description: {product_data[2]}")
                print(f"Price: {product_data[3]}")
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def IsProductInStock(self, product_id):
        try:
            if not isinstance(product_id, int) or product_id < 0:
                raise InvalidIDError()

            self.open()
            query = '''
                SELECT Inventory.QuantityInStock
                FROM Products
                INNER JOIN Inventory ON Products.Product_ID = Inventory.ProductID
                WHERE Products.Product_ID = %s
            '''

            self.stmt.execute(query, (product_id,))
            result = self.stmt.fetchone()

            if result and result[0] > 0:
                print("Product is in stock:")
                return result
            else:
                return False

        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")
            return False
```

GetProductDetails O/P:



```
1.Add Product   2.View Product Details  3.Update Product Details
4.Delete Product    5.Get Productby ID  6.Check for availability
7.Exit
enter your choice2
--Database Is Connected--

_________________Records In Products Table______________________
(1, 'Laptop', 'High-performance laptop with SSD', 1099)
(2, 'Camera', 'Latest model with dual camera', 769)
(3, 'Tablet', '10-inch tablet with touchscreen', 329)
(4, 'Camera', 'Fitness tracker with heart rate monitor', 219)
(5, 'Headphones', 'Noise-canceling wireless headphones', 164)
(6, 'Camera', 'DSLR camera with 18-55mm lens', 879)
(7, 'TV', '4K Ultra HD smart TV', 1429)
(8, 'Speaker', 'Bluetooth portable speaker', 87)
(9, 'Gaming Console', 'Next-gen gaming console', 549)
(10, 'Router', 'High-speed Wi-Fi router', 142)
(11, 'Smart Speaker', 'Voice-controlled smart speaker', 749)
(20, 'Earbuds', 'Wireless earphones', 500)
Connection Closed.
```

UpdateProductInfo O/P:



```
Input Product ID to be Updated: 20
Enter Product Name ((Press Enter to skip)):
Enter Description:(Press Enter to skip)
Enter Price: (Press Enter to skip): 650
--Database Is Connected--
Record updated successfully.
--Database Is Connected--

_________________Records In Products Table______________________
(1, 'Laptop', 'High-performance laptop with SSD', 1099)
(2, 'Camera', 'Latest model with dual camera', 769)
(3, 'Tablet', '10-inch tablet with touchscreen', 329)
(4, 'Camera', 'Fitness tracker with heart rate monitor', 219)
(5, 'Headphones', 'Noise-canceling wireless headphones', 164)
(6, 'Camera', 'DSLR camera with 18-55mm lens', 879)
(7, 'TV', '4K Ultra HD smart TV', 1429)
(8, 'Speaker', 'Bluetooth portable speaker', 87)
(9, 'Gaming Console', 'Next-gen gaming console', 549)
(10, 'Router', 'High-speed Wi-Fi router', 142)
(11, 'Smart Speaker', 'Voice-controlled smart speaker', 749)
(20, 'Earbuds', 'Wireless earphones', 650)
Connection Closed.
```

Orders Class:

Attributes:

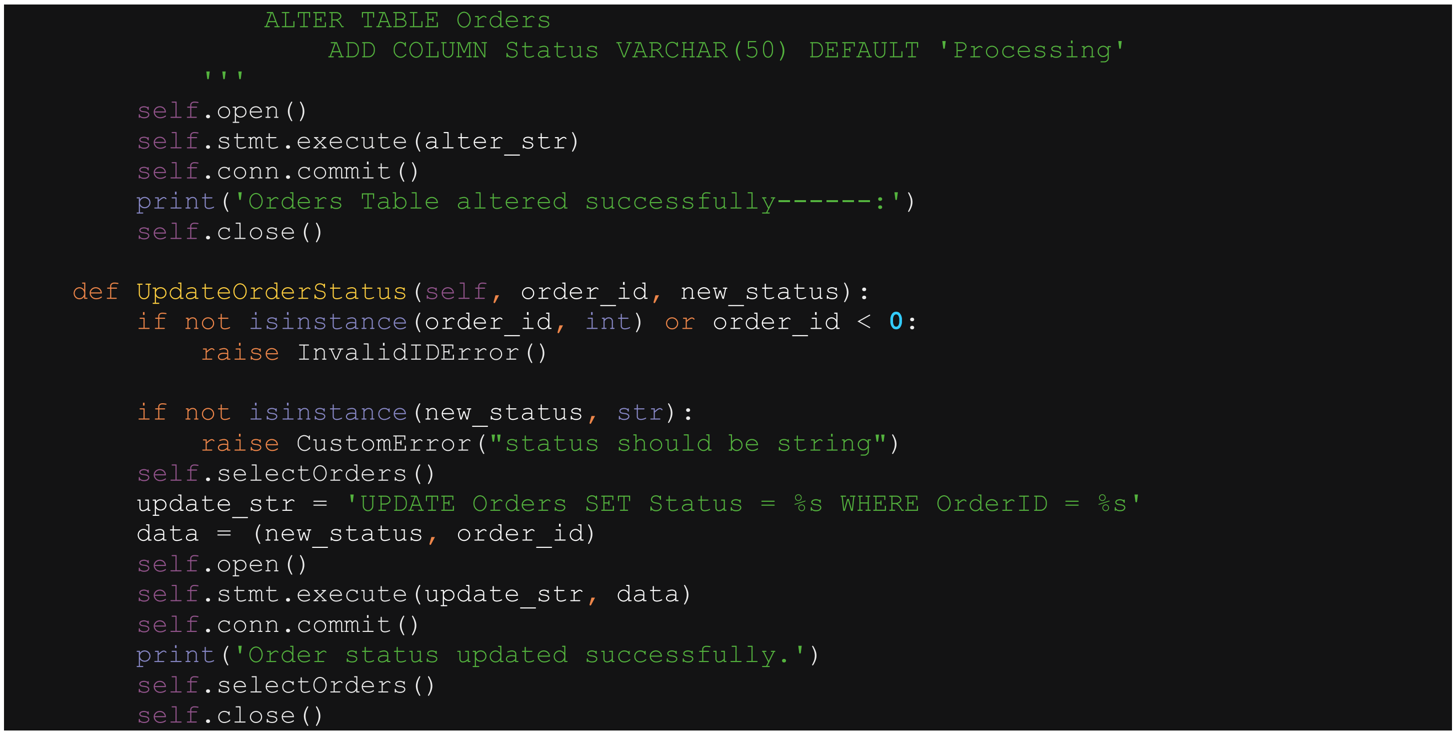
- OrderID (int)
- Customer (Customer) - Use composition to reference the Customer who placed the order.
- OrderDate (DateTime)
- TotalAmount (decimal)

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.

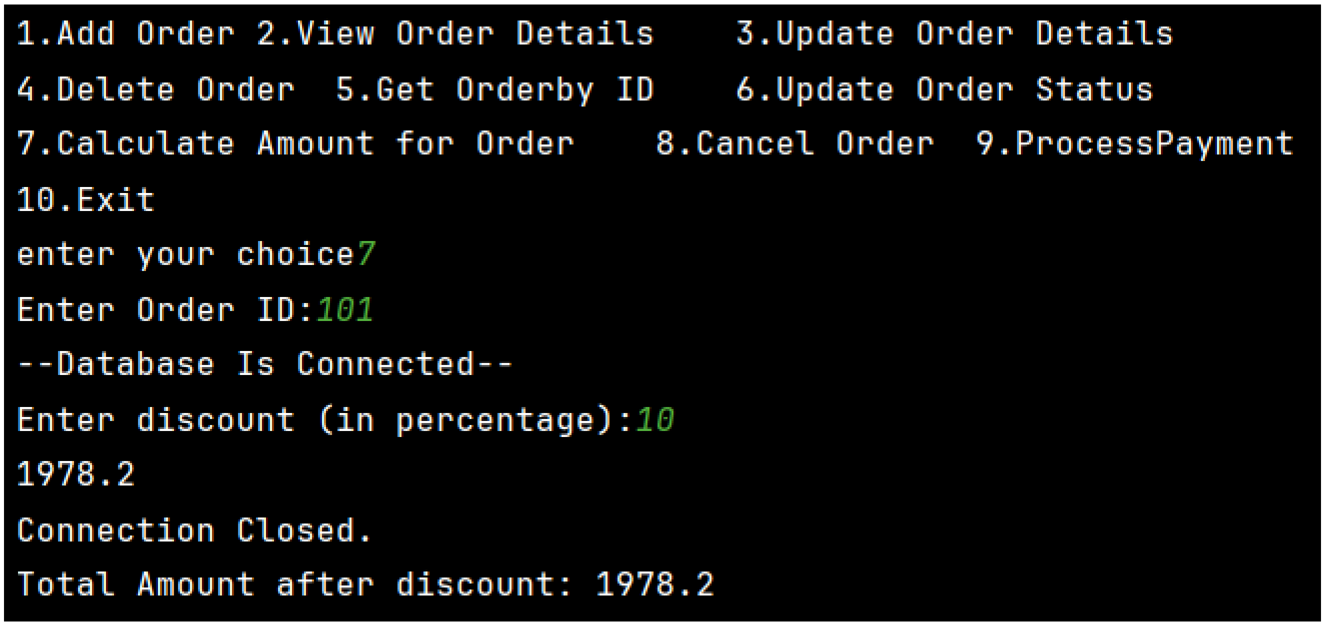
```python
from exceptions.InvalidIDError import InvalidIDError
from exceptions.InvalidPriceError import InvalidPriceError
from exceptions.CustomError import CustomError
from util.DBConnUtil import dbConnection
from datetime import datetime
from decimal import Decimal
from exceptions.PaymentFailedException import  PaymentFailedException
from exceptions.InvalidNameError import InvalidNameError
class Orders(dbConnection):

    def create(self):
        create_orders_str = '''
        CREATE TABLE IF NOT EXISTS Orders (
            OrderID INT PRIMARY KEY AUTO_INCREMENT,
            CustomerID INT,
            OrderDate DATE,
            TotalAmount DECIMAL(7, 2),
            FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE
        );
        '''
        self.open()
        self.stmt.execute(create_orders_str)
        self.stmt.close()
        print('Orders Tables created successfully------:')
        return f'Created succesfully'

    def addOrder(self):
        Order_ID = int(input('Enter OrderID:'))
        if not isinstance(Order_ID, int) or Order_ID < 0:
            raise InvalidIDError()
        self.Order_Id = Order_ID

        CustomerID = int(input('Enter CustomerID:'))
        if not isinstance(CustomerID, int) or CustomerID < 0:
            raise InvalidIDError()
        self.CustomerId = CustomerID
        order_date_input = input('Enter Order Date (YYYY-MM-DD) or leave blank for
current date: ')

        TotalAmount = int(input('Enter TotalAmount:'))
        if not isinstance(TotalAmount, (int, float)) or TotalAmount<0:
            raise InvalidPriceError()
        self.TotalAmount = TotalAmount

        Status = input('Enter status:')
        if not isinstance(Status, str):
            raise InvalidIDError()
        self.Status = Status

        if not order_date_input:
            order_date = datetime.now().strftime('%Y-%m-%d')
        else:
            order_date = order_date_input

        data = [(self.Order_Id, self.CustomerId, order_date, self.TotalAmount,
self.Status)]
        insert_order_str = '''INSERT INTO Orders (OrderId,CustomerID, OrderDate,
TotalAmount,Status)
                              VALUES (%s,%s,%s,%s, %s)'''

        self.open()
        self.stmt.executemany(insert_order_str, data)
        self.conn.commit()
        print('Order Record Inserted Successfully..')
        self.close()

    def GetOrderDetails(self, order_id):
        try:
            if not isinstance(order_id, int) or order_id < 0:
                raise InvalidIDError()
            query = '''
                SELECT OrderID, CustomerID, OrderDate, TotalAmount, Status
                FROM Orders
                WHERE OrderID = %s
            '''

            self.open()
            self.stmt.execute(query, (order_id,))
            order_details = self.stmt.fetchone()

            if order_details:
                print('\nOrder Details:')
                print(f"OrderID: {order_details[0]}")
                print(f"CustomerID: {order_details[1]}")
                print(f"OrderDate: {order_details[2]}")
                print(f"TotalAmount: {order_details[3]}")
                print(f"Status: {order_details[4]}")
            else:
                print(f'Order with OrderID {order_id} not found.')

        except ValueError as ve:
            print(f"Error: {ve}")

        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

        finally:
            self.close()
```

```python
    def selectOrders(self):
        self.open()
        select_orders_str = '''SELECT * FROM Orders'''
        self.stmt.execute(select_orders_str)
        records = self.stmt.fetchall()
        print('')
        print('_________________Records In Orders Table________________________')
        for record in records:
            print(record)
        self.close()

    def updateOrder(self):
        self.selectOrders()
        order = int(input('Input Order ID to be Updated: '))
        if not isinstance(order, int) or order < 0:
            raise InvalidIDError()
        orderId = order
        update_order_str = 'UPDATE Orders SET '
        data = []

        self.customerID = input('Enter Customer ID (Press Enter to skip): ')
        if self.customerID:
            update_order_str += 'CustomerID=%s, '
            data.append(self.customerID)

        self.orderDate = input('Enter Order Date (YYYY-MM-DD) (Press Enter to skip): ')
        if self.orderDate:
            update_order_str += 'OrderDate=%s, '
            data.append(self.orderDate)

        self.totalAmount = input('Enter Total Amount (Press Enter to skip): ')
        if self.totalAmount:
            update_order_str += 'TotalAmount=%s, '
            data.append(self.totalAmount)

        self.Status = input('Enter Status of Order (Press Enter to skip): ')
        if self.Status:
            update_order_str += 'Status=%s, '
            data.append(self.Status)

        update_order_str = update_order_str.rstrip(', ')

        update_order_str += ' WHERE OrderID=%s'
        data.append(orderId)

        self.open()
        self.stmt.execute(update_order_str, data)
        self.conn.commit()
        print('Order Record updated successfully.')
        self.selectOrders()

    def deleteOrder(self):
        self.selectOrders()
        Order_id = int(input('Input Order ID to be Deleted: '))
        if not isinstance(Order_id, int) or Order_id < 0:
            raise InvalidIDError()
        Id = Order_id
        delete_order_str = f'DELETE FROM Orders WHERE OrderID={Id}'
        self.open()
        self.stmt.execute(delete_order_str)
        self.conn.commit()
        print('Order Record Deleted Successfully..')
        self.selectOrders()
    def CalculateTotalAmount(self):
        try:
            OrderID = int(input('Enter Order ID:'))
            if not isinstance(OrderID, int) or OrderID < 0:
                raise InvalidIDError()

            self.OrderID = OrderID
            self.open()

            statement = '''
                SELECT Price, Quantity
                FROM OrderDetails
                INNER JOIN Orders ON Orders.OrderID = OrderDetails.OrderID
                INNER JOIN Products ON Products.Product_ID = OrderDetails.ProductID
                WHERE Orders.OrderID = %s
            '''

            self.stmt.execute(statement, (OrderID,))
            records = self.stmt.fetchall()

            if not records:
                raise CustomError("No records found for the specified Order ID.")

            total_amount = 0

            for record in records:
                price = float(record[0])
                quantity = int(record[1])
                total_amount += price * quantity

            discount = float(input("Enter discount (in percentage):"))

            if discount < 0 or discount > 100:
                raise CustomError("discount should be between 0-100")

            discount /= 100
            total_amount *= (1 - discount)
            print(total_amount)

            update_statement = 'UPDATE Orders SET TotalAmount=%s WHERE OrderID=%s'
            update_data = (Decimal(total_amount), OrderID)

            self.stmt.execute(update_statement, update_data)
            self.conn.commit()
            self.close()
            print("Total Amount after discount:", total_amount)

        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def CancelOrder(self, order_id):
        try:
            if not isinstance(order_id, int) or order_id < 0:
                raise InvalidIDError()

            query = '''
                SELECT OrderDetails.Quantity, Inventory.QuantityInStock,
Inventory.ProductID
                FROM Orders
                JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
                JOIN Inventory ON OrderDetails.ProductID = Inventory.ProductID
                WHERE Orders.OrderID = %s
            '''
            self.open()
            self.stmt.execute(query, (order_id,))
            result = self.stmt.fetchall()
            if result:
                print('\nUpdating Inventory.QuantityInStock:')
                for row in result:
                    order_quantity = row[0]
                    inventory_quantity = row[1]
                    product_id = row[2]

                    new_quantity_in_stock = inventory_quantity + order_quantity
                    update_query = 'UPDATE Inventory SET QuantityInStock = %s WHERE
ProductID = %s'
                    update_data = (new_quantity_in_stock, product_id)
                    self.stmt.execute(update_query, update_data)

                delete_order_str = f'DELETE FROM Orders WHERE OrderID={order_id}'
                self.stmt.execute(delete_order_str)
                self.conn.commit()
                self.close()
                print(f'Order with OrderID {order_id} canceled successfully.')


            else:
                print(f'No data found for OrderID {order_id}.')

        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def ProcessPayment(self,oid):


        CustomerID = int(input('Enter CustomerID:'))
        if not isinstance(CustomerID, int) or CustomerID < 0:
            raise InvalidIDError()
        self.CustomerId = CustomerID


        self.open()
        TotalAmount="SELECT TotalAmount from Orders where customerid=%s"

        self.stmt.execute(TotalAmount, (CustomerID,))
        records = self.stmt.fetchone()
        rec=list(records)
        totalamount=rec[0]
        try:

            amount = totalamount
            entered_amount = float(input("enter amount you want to pay"))
            if (entered_amount == amount):
                print("Payment processed successfully.")
            else:

                raise PaymentFailedException()
                self.ProcessPayment(oid)
        except PaymentFailedException as e:

            print(f"Error processing payment: {e}")


    def AlterTable(self):
        alter_str = '''
```



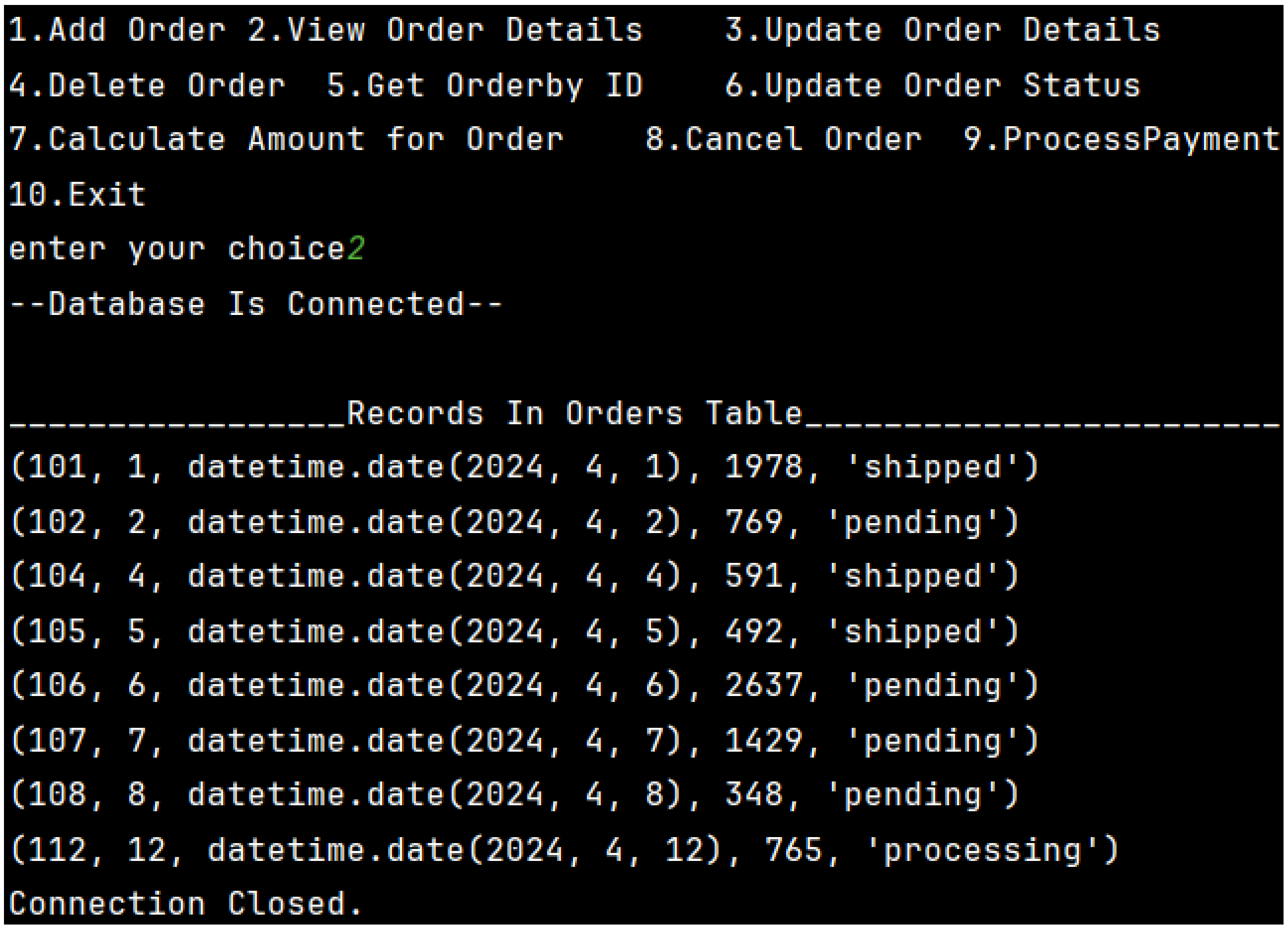
```
                ALTER TABLE Orders
                    ADD COLUMN Status VARCHAR(50) DEFAULT 'Processing'
            '''
        self.open()
        self.stmt.execute(alter_str)
        self.conn.commit()
        print('Orders Table altered successfully------:')
        self.close()

    def UpdateOrderStatus(self, order_id, new_status):
        if not isinstance(order_id, int) or order_id < 0:
            raise InvalidIDError()

        if not isinstance(new_status, str):
            raise CustomError("status should be string")
        self.selectOrders()
        update_str = 'UPDATE Orders SET Status = %s WHERE OrderID = %s'
        data = (new_status, order_id)
        self.open()
        self.stmt.execute(update_str, data)
        self.conn.commit()
        print('Order status updated successfully.')
        self.selectOrders()
        self.close()
```

CalculateTotalAmount O/P:



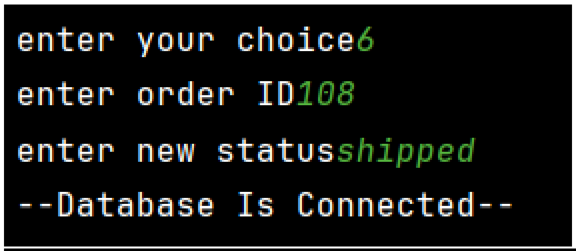
```
1.Add Order 2.View Order Details    3.Update Order Details
4.Delete Order  5.Get Orderby ID    6.Update Order Status
7.Calculate Amount for Order    8.Cancel Order  9.ProcessPayment
10.Exit
enter your choice7
Enter Order ID:101
--Database Is Connected--
Enter discount (in percentage):10
1978.2
Connection Closed.
Total Amount after discount: 1978.2
```

GetOrderDetails O/P:

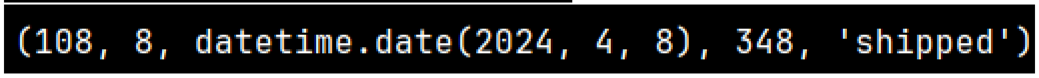


```
1.Add Order 2.View Order Details    3.Update Order Details
4.Delete Order  5.Get Orderby ID    6.Update Order Status
7.Calculate Amount for Order    8.Cancel Order  9.ProcessPayment
10.Exit
enter your choice2
--Database Is Connected--

_________________Records In Orders Table________________________
(101, 1, datetime.date(2024, 4, 1), 1978, 'shipped')
(102, 2, datetime.date(2024, 4, 2), 769, 'pending')
(104, 4, datetime.date(2024, 4, 4), 591, 'shipped')
(105, 5, datetime.date(2024, 4, 5), 492, 'shipped')
(106, 6, datetime.date(2024, 4, 6), 2637, 'pending')
(107, 7, datetime.date(2024, 4, 7), 1429, 'pending')
(108, 8, datetime.date(2024, 4, 8), 348, 'pending')
(112, 12, datetime.date(2024, 4, 12), 765, 'processing')
Connection Closed.
```

UpdateOrderStatus O/P:

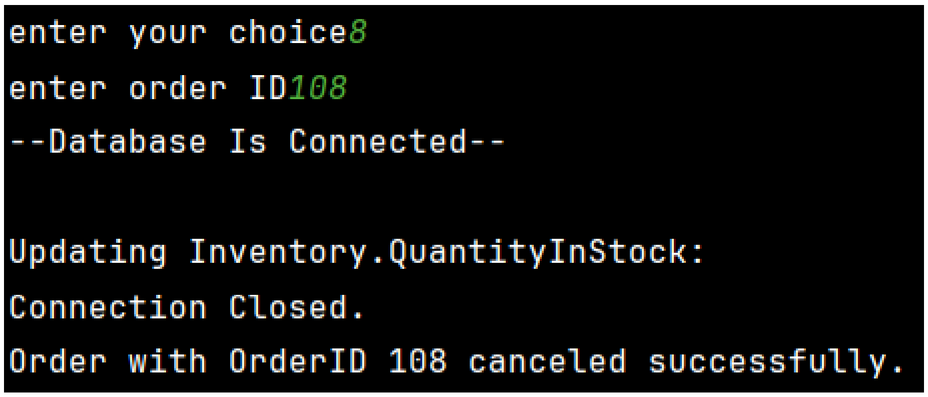


```
enter your choice6
enter order ID108
enter new statusshipped
--Database Is Connected--
```



```
(108, 8, datetime.date(2024, 4, 8), 348, 'shipped')
```

CancelOrder O/P:



```
enter your choice8
enter order ID108
--Database Is Connected--

Updating Inventory.QuantityInStock:
Connection Closed.
Order with OrderID 108 canceled successfully.
```

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)

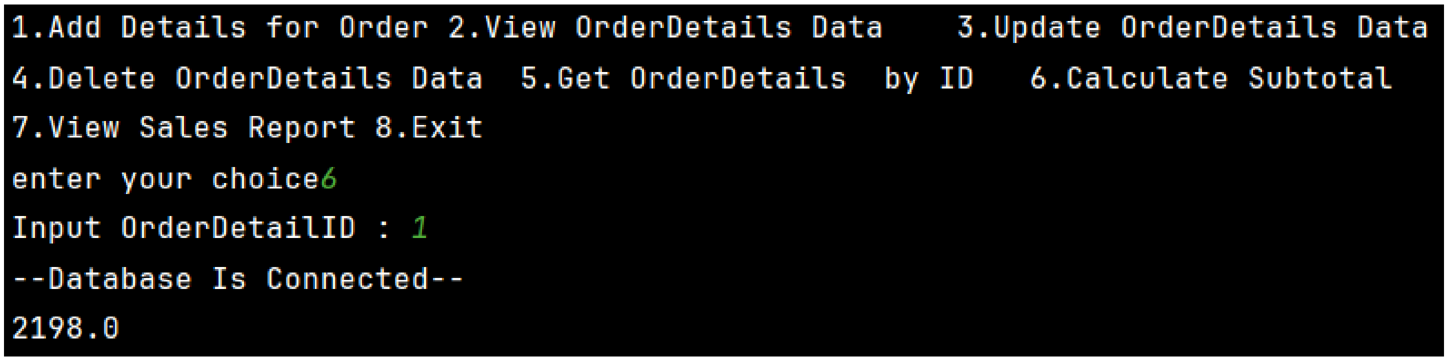
Methods:

- 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.

```
from exceptions.CustomError import CustomError
from exceptions.InvalidIDError import InvalidIDError
from exceptions.InvalidQuantityError import InvalidQuantityError
from exceptions.IncompleteOrderException import IncompleteOrderException
from util.DBConnUtil import dbConnection
import decimal

class OrderDetails(dbConnection):

    def create(self):
        create_orderdetails_str = '''
        CREATE TABLE IF NOT EXISTS OrderDetails (
            OrderDetailID INT PRIMARY KEY AUTO_INCREMENT,
            OrderID INT,
            ProductID INT,
            Quantity INT,
            FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE,
            FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE CASCADE
        );
        '''
        self.open()
        self.stmt.execute(create_orderdetails_str)
        self.stmt.close()
        print('OrderDetails Tables created successfully------:')

    def addOrderDetail(self):
        OrderDetailID = int(input('Enter Orderdetail ID:'))
        if not isinstance(OrderDetailID, int) or OrderDetailID < 0:
            raise InvalidIDError()
        self.OrderDetailID = OrderDetailID

        OrderID = int(input('Enter Order ID:'))
        if not isinstance(OrderID, int) or OrderID < 0:
            raise InvalidIDError()
        self.OrderID = OrderID

        ProductID = int(input('Enter Product ID:'))
        if not isinstance(ProductID, int):
            raise InvalidIDError()
        self.ProductID = ProductID
        Quantity = int(input('Enter Quantity:'))
        if not isinstance(Quantity, int) or Quantity<0:
            raise InvalidQuantityError()
        self.Quantity = Quantity

        data = [(self.OrderDetailID,self.OrderID, self.ProductID, self.Quantity)]
        insert_orderdetail_str = '''INSERT INTO OrderDetails (OrderDetailID, OrderID, 
ProductID, Quantity)
                                    VALUES (%s, %s, %s, %s)'''

        self.open()
        self.stmt.executemany(insert_orderdetail_str, data)
        self.conn.commit()
        print('OrderDetail Record Inserted Successfully..')
        self.close()

    def selectOrderDetails(self):
        self.open()
        select_orderdetails_str = '''SELECT * FROM OrderDetails'''
        self.stmt.execute(select_orderdetails_str)
        records = self.stmt.fetchall()
        print('')
        print('_________________Records In OrderDetails Table________________________')
        for record in records:
            print(record)
        self.close()

    def updateOrderDetail(self):
        self.selectOrderDetails()
        order = int(input('Input OrderDetail ID to be Updated: '))
        if not isinstance(order, int) or order < 0:
            raise InvalidIDError()
        orderDetailId = order
        update_orderdetail_str = 'UPDATE OrderDetails SET '
        data = []

        self.orderID = int(input('Enter Order ID (Press Enter to skip): '))
        if self.orderID:

            update_orderdetail_str += 'OrderID=%s, '
            data.append(self.orderID)

        self.productID = int(input('Enter Product ID (Press Enter to skip): '))
        if self.productID:

            update_orderdetail_str += 'ProductID=%s, '
            data.append(self.productID)

        self.quantity = int(input('Enter Quantity (Press Enter to skip): '))
        if self.quantity:

            update_orderdetail_str += 'Quantity=%s, '
            data.append(self.quantity)

        update_orderdetail_str = update_orderdetail_str.rstrip(', ')

        update_orderdetail_str += ' WHERE OrderDetailID=%s'
        data.append(orderDetailId)

        self.open()
        self.stmt.execute(update_orderdetail_str, data)
        self.conn.commit()
        print('OrderDetail Record updated successfully.')
```

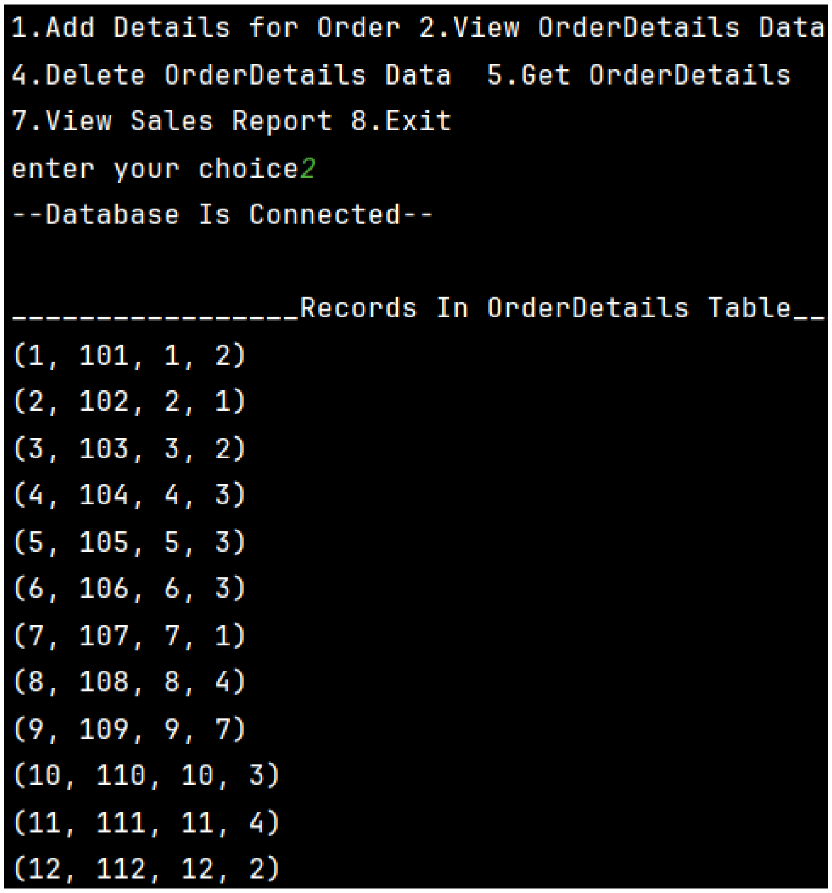
```python
        self.selectOrderDetails()

    def deleteOrderDetail(self):
        OrderDetail_id = int(input('Input OrderDetail ID to be Deleted: '))
        if not isinstance(OrderDetail_id, int):
            raise InvalidIDError()
        else:
            if OrderDetail_id < 0:
                raise InvalidIDError()
        Id = OrderDetail_id
        delete_orderdetail_str = f'DELETE FROM OrderDetails WHERE OrderDetailID={Id}'
        self.open()
        self.stmt.execute(delete_orderdetail_str)
        self.conn.commit()
        print('OrderDetail Record Deleted Successfully..')

    def GetOrderDetails(self, order_id):
        try:
            if not isinstance(order_id, int) or order_id < 0:
                raise InvalidIDError()

            query = '''
                SELECT *
                FROM OrderDetails join Products on
                OrderDetails.productid = products.product_id
                WHERE OrderDetailID=%s
            '''

            self.open()
            self.stmt.execute(query, (order_id,))
            order_details = self.stmt.fetchall()
            self.close()

            if order_details:
                print('\nOrder Details:')
                for detail in order_details:
                    print(f"OrderDetailID: {detail[0]}")
                    print(f"OrderID: {detail[1]}")
                    print(f"ProductID: {detail[2]}")
                    print(f"Quantity: {detail[3]}")
            else:
                raise IncompleteOrderException()

        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def CalculateSubtotal(self):
        orderDetailId = int(input('Input OrderDetailID : '))
        if orderDetailId < 0 or orderDetailId < 0:
            raise InvalidIDError()
        self.open()
        statement = '''select  Price,Quantity from OrderDetails
        inner join Products on Products.Product_ID = OrderDetails.ProductID
        where OrderDetails.OrderDetailID = %s
        '''
        self.stmt.execute(statement, (orderDetailId,))
        records = self.stmt.fetchone()
        print(float(records[0]) * records[1])
        self.close()

    def AddDiscount(self, dis):
        if not isinstance(dis, (int, float)) or dis < 0 or dis > 100:
            raise CustomError("discount should be between 0-100")
        self.discount = dis

    def SalesReporting(self):
        self.open()
        select_orderdetails_str = '''SELECT ProductID,SUM(Quantity) FROM OrderDetails
group by ProductID'''
        self.stmt.execute(select_orderdetails_str)
        records = self.stmt.fetchall()
        print('')
        print('_________________Sales Report________________________')
        for record in records:
            print(f"Product ID is {record[0]} sold {int(record[1])} units")
        self.close()
```

CalculateSubtotal O/P:



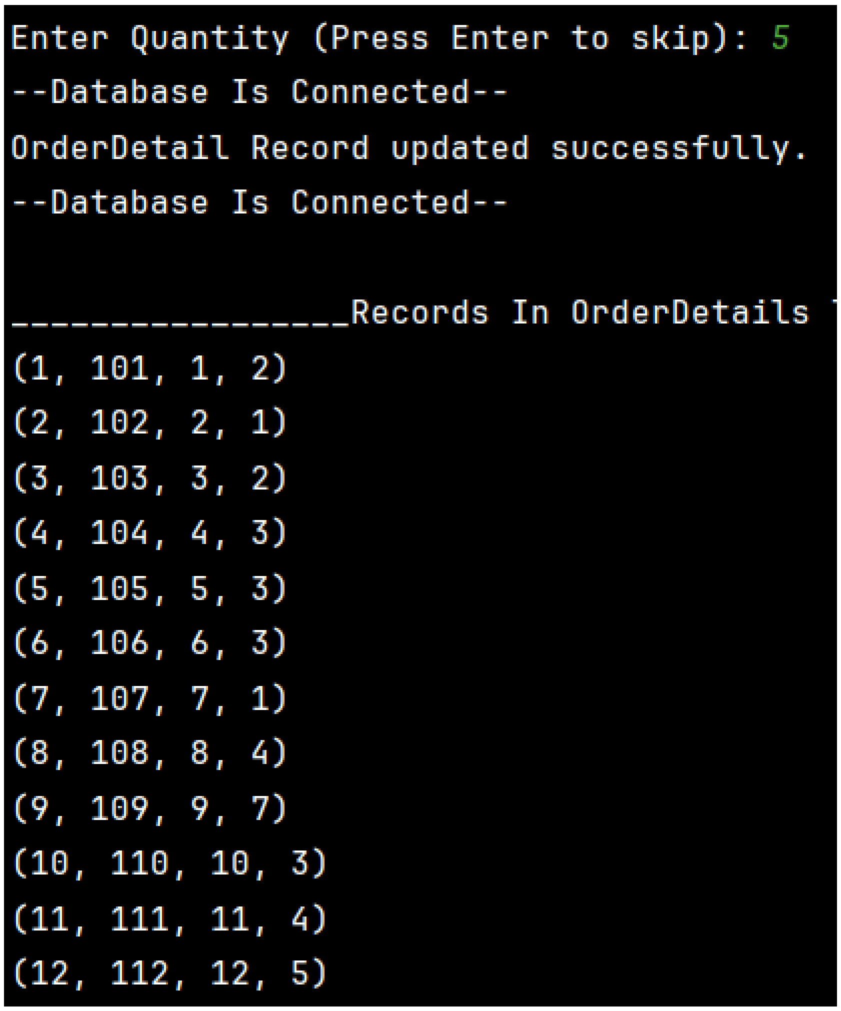
```
1.Add Details for Order 2.View OrderDetails Data    3.Update OrderDetails Data
4.Delete OrderDetails Data  5.Get OrderDetails  by ID   6.Calculate Subtotal
7.View Sales Report 8.Exit
enter your choice6
Input OrderDetailID : 1
--Database Is Connected--
2198.0
```

GetOrderDetailInfo O/P:



```
1.Add Details for Order 2.View OrderDetails Data
4.Delete OrderDetails Data  5.Get OrderDetails
7.View Sales Report 8.Exit
enter your choice2
--Database Is Connected--

_________________Records In OrderDetails Table__
(1, 101, 1, 2)
(2, 102, 2, 1)
(3, 103, 3, 2)
(4, 104, 4, 3)
(5, 105, 5, 3)
(6, 106, 6, 3)
(7, 107, 7, 1)
(8, 108, 8, 4)
(9, 109, 9, 7)
(10, 110, 10, 3)
(11, 111, 11, 4)
(12, 112, 12, 2)
```

UpdateQuantity O/P:



```
Enter Quantity (Press Enter to skip): 5
--Database Is Connected--
OrderDetail Record updated successfully.
--Database Is Connected--


_________________Records In OrderDetails
(1, 101, 1, 2)
(2, 102, 2, 1)
(3, 103, 3, 2)
(4, 104, 4, 3)
(5, 105, 5, 3)
(6, 106, 6, 3)
(7, 107, 7, 1)
(8, 108, 8, 4)
(9, 109, 9, 7)
(10, 110, 10, 3)
(11, 111, 11, 4)
(12, 112, 12, 5)
```

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.

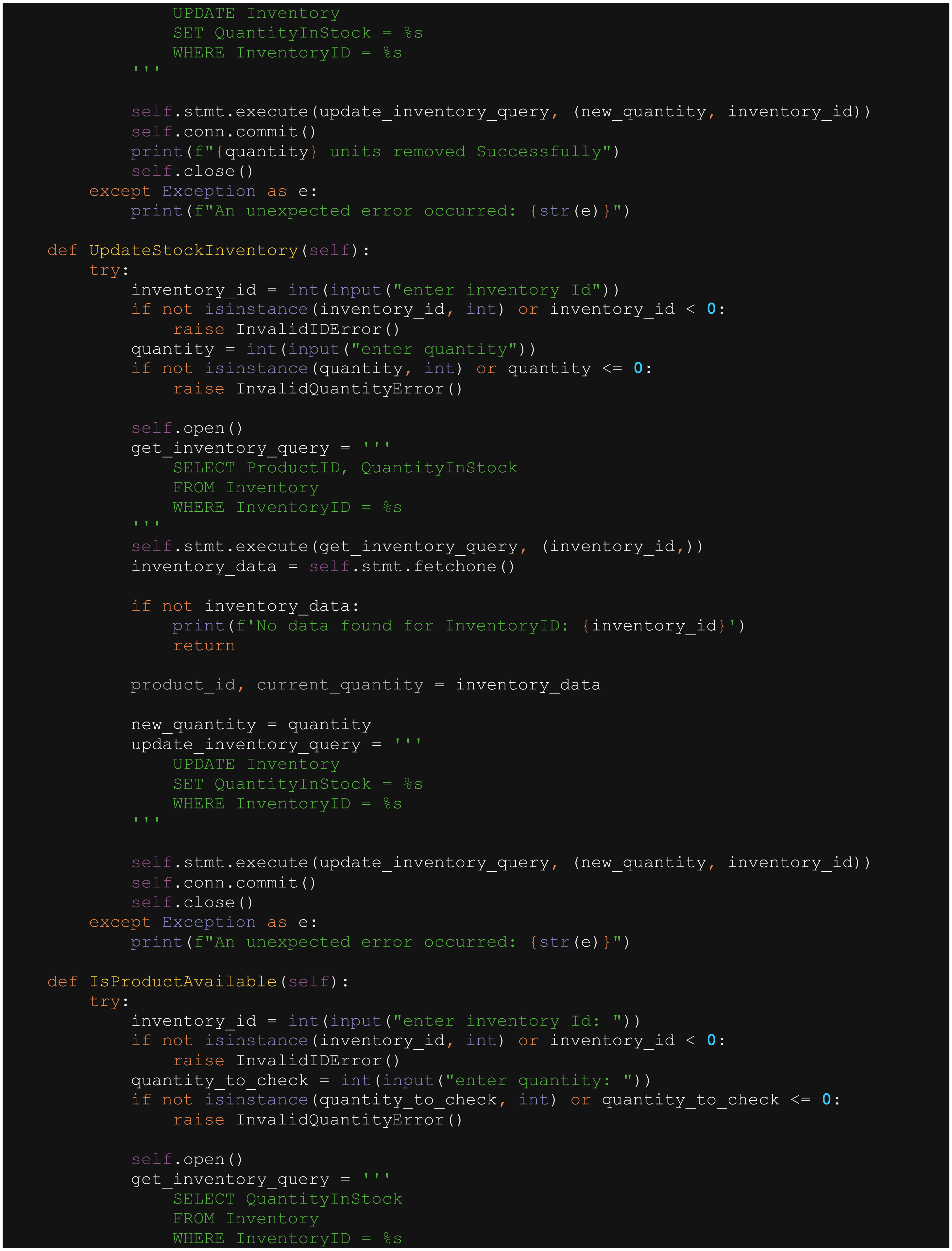
• ListLowStockProducts(int threshold): A method to list products with quantities below a specified threshold, indicating low stock.

• 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.

```
from exceptions.InvalidIDError import InvalidIDError
from exceptions.InvalidQuantityError import InvalidQuantityError
from util.DBConnUtil import dbConnection
from exceptions.InsufficientStockException import InsufficientStockException
from datetime import datetime


class Inventory(dbConnection):

    def create(self):
        create_inventory_str = '''
        CREATE TABLE IF NOT EXISTS Inventory (
            InventoryID INT PRIMARY KEY AUTO_INCREMENT,
            ProductID INT,
            QuantityInStock INT,
            LastStockUpdate DATE,
            FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE CASCADE
        );
        '''
        self.open()
        self.stmt.execute(create_inventory_str)
        self.stmt.close()
        print('Inventory Tables created successfully------:')

    def addInventory(self):
        ProductID = int(input('Enter Product ID:'))
        if not isinstance(ProductID, int) or ProductID < 0:
            raise InvalidIDError()
        self.productID = ProductID

        QuantityInStock = int(input('Enter Quantity in stock:'))
        if not isinstance(QuantityInStock, int)or QuantityInStock<0:
            raise InvalidQuantityError()
        self.quantityInStock = QuantityInStock

        last_Stock_Update = input("enter date yyyy-mm-dd:")
        if not last_Stock_Update:
            self.lastStockUpdate = datetime.now().strftime('%Y-%m-%d')
        self.lastStockUpdate = last_Stock_Update
        data = [(self.productID, self.quantityInStock, self.lastStockUpdate)]
        insert_inventory_str = '''INSERT INTO Inventory (ProductID, QuantityInStock,
LastStockUpdate)
                                  VALUES (%s, %s, %s)'''

        self.open()
        self.stmt.executemany(insert_inventory_str, data)
        self.conn.commit()
```

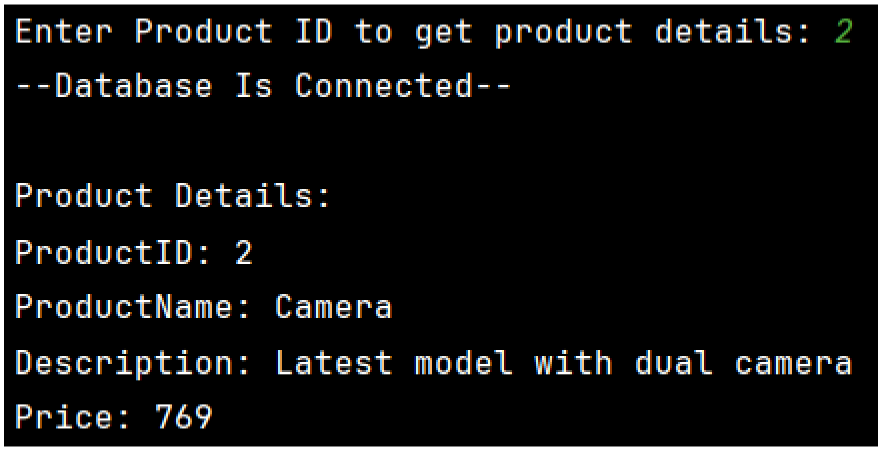
```python
        print('Inventory Record Inserted Successfully..')
        self.close()
    def selectInventory(self):
        self.open()
        select_inventory_str = '''SELECT * FROM Inventory'''
        self.stmt.execute(select_inventory_str)
        records = self.stmt.fetchall()
        print('')
        print('_________________Records In Inventory Table________________________')
        for record in records:
            print(record)
        self.close()

    def updateInventory(self):
        self.selectInventory()
        Inventory_id = int(input('Input Inventory ID to be Updated: '))
        if not isinstance(Inventory_id, int) or Inventory_id < 0:
            raise InvalidIDError()
        inventoryId = Inventory_id
        update_inventory_str = 'UPDATE Inventory SET '
        data = []

        self.productID = int(input('Enter Product ID (Press Enter to skip): '))
        if self.productID:
            update_inventory_str += 'ProductID=%s, '
            data.append(self.productID)

        self.quantityInStock = int(input('Enter Quantity in Stock (Press Enter to skip):
'))
        if self.quantityInStock:
            update_inventory_str += 'QuantityInStock=%s, '
            data.append(self.quantityInStock)

        self.lastStockUpdate = input('Enter Last Stock Update (YYYY-MM-DD HH:MM:SS)
(Press Enter to skip): ')
        if self.lastStockUpdate:
            update_inventory_str += 'LastStockUpdate=%s, '
            data.append(self.lastStockUpdate)

        update_inventory_str = update_inventory_str.rstrip(', ')

        update_inventory_str += ' WHERE InventoryID=%s'
        data.append(inventoryId)

        self.open()
        self.stmt.execute(update_inventory_str, data)
        self.conn.commit()
        print('Inventory Record updated successfully.')
        self.selectInventory()

    def deleteInventory(self):
        Inventory_id = int(input('Input Inventory ID to be Deleted: '))
        if not isinstance(Inventory_id, int):
            raise InvalidIDError()
        else:
            if Inventory_id < 0:
                raise InvalidIDError()
        inventoryId = Inventory_id
        delete_inventory_str = f'DELETE FROM Inventory WHERE InventoryID={inventoryId}'
        self.open()
        self.stmt.execute(delete_inventory_str)
        self.conn.commit()
        print('Inventory Record Deleted Successfully..')
    def GetProduct(self):
        try:
            product_id = int(input('Enter Product ID to get product details: '))
            if not isinstance(product_id, int) or product_id < 0:
                raise InvalidIDError()

            self.open()
            query = '''
                SELECT Products.*
                FROM Products
                INNER JOIN Inventory ON Products.Product_ID = Inventory.ProductID
                WHERE Inventory.ProductID = %s
            '''

            self.stmt.execute(query, (product_id,))
            product_data = self.stmt.fetchone()

            if product_data:
                print('\nProduct Details:')
                print(f"ProductID: {product_data[0]}")
                print(f"ProductName: {product_data[1]}")
                print(f"Description: {product_data[2]}")
                print(f"Price: {product_data[3]}")
            else:
                print(f'No product found with ProductID: {product_id}')
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def GetQuantityInStock(self):
        try:
            inventory_id = int(input("enter inventiry number:"))
            if not isinstance(inventory_id, int) or inventory_id < 0:
                raise InvalidIDError()

            self.open()
            query = '''
                SELECT QuantityInStock
                FROM Inventory
                WHERE InventoryID = %s
            '''

            self.stmt.execute(query, (inventory_id,))
            quantity_in_stock = self.stmt.fetchone()

            if quantity_in_stock:
                print(f'Quantity in Stock for InventoryID {inventory_id}:
{quantity_in_stock[0]}')
            else:
                print(f'No data found for InventoryID: {inventory_id}')
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def AddToInventory(self):
        try:
            inventory_id = int(input("enter inventory Id:"))
            if not isinstance(inventory_id, int) or inventory_id < 0:
                raise InvalidIDError()
            quantity = int(input("enter quantity to be added: "))
            if not isinstance(quantity, int) or quantity <= 0:
                raise InvalidQuantityError()
            self.open()
            get_inventory_query = '''
                SELECT ProductID, QuantityInStock
                FROM Inventory
                WHERE InventoryID = %s
            '''
            self.stmt.execute(get_inventory_query, (inventory_id,))
            inventory_data = self.stmt.fetchone()

            if not inventory_data:
                print(f'No data found for InventoryID: {inventory_id}')
                return

            product_id, current_quantity = inventory_data

            new_quantity = current_quantity + quantity

            update_inventory_query = '''
                UPDATE Inventory
                SET QuantityInStock = %s
                WHERE InventoryID = %s
            '''

            self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
            print(f"{quantity} units added Successfully")
            self.conn.commit()
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def RemoveFromInventory(self):
        try:
            inventory_id = int(input("enter inventory Id: "))
            if not isinstance(inventory_id, int) or inventory_id < 0:
                raise InvalidIDError()
            self.inventory_id = inventory_id

            quantity = int(input("enter quantity to be removed : "))
            if not isinstance(quantity, int) or quantity <= 0:
                raise InvalidQuantityError()
            self.quantity = quantity

            self.open()
            get_inventory_query = '''
                SELECT ProductID, QuantityInStock
                FROM Inventory
                WHERE InventoryID = %s
            '''
            self.stmt.execute(get_inventory_query, (inventory_id,))
            inventory_data = self.stmt.fetchone()

            if not inventory_data:
                print(f'No data found for InventoryID: {inventory_id}')
                return

            product_id, current_quantity = inventory_data

            new_quantity = current_quantity - quantity
            if new_quantity < 0:
                raise InsufficientStockException()

            update_inventory_query = '''
```



```
                UPDATE Inventory
                SET QuantityInStock = %s
                WHERE InventoryID = %s
            '''

            self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
            self.conn.commit()
            print(f"{quantity} units removed Successfully")
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def UpdateStockInventory(self):
        try:
            inventory_id = int(input("enter inventory Id"))
            if not isinstance(inventory_id, int) or inventory_id < 0:
                raise InvalidIDError()
            quantity = int(input("enter quantity"))
            if not isinstance(quantity, int) or quantity <= 0:
                raise InvalidQuantityError()

            self.open()
            get_inventory_query = '''
                SELECT ProductID, QuantityInStock
                FROM Inventory
                WHERE InventoryID = %s
            '''
            self.stmt.execute(get_inventory_query, (inventory_id,))
            inventory_data = self.stmt.fetchone()

            if not inventory_data:
                print(f'No data found for InventoryID: {inventory_id}')
                return

            product_id, current_quantity = inventory_data

            new_quantity = quantity
            update_inventory_query = '''
                UPDATE Inventory
                SET QuantityInStock = %s
                WHERE InventoryID = %s
            '''

            self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
            self.conn.commit()
            self.close()
        except Exception as e:
            print(f"An unexpected error occurred: {str(e)}")

    def IsProductAvailable(self):
        try:
            inventory_id = int(input("enter inventory Id: "))
            if not isinstance(inventory_id, int) or inventory_id < 0:
                raise InvalidIDError()
            quantity_to_check = int(input("enter quantity: "))
            if not isinstance(quantity_to_check, int) or quantity_to_check <= 0:
                raise InvalidQuantityError()

            self.open()
            get_inventory_query = '''
                SELECT QuantityInStock
                FROM Inventory
                WHERE InventoryID = %s
            '''
            self.stmt.execute(get_inventory_query, (inventory_id,))
            current_quantity = self.stmt.fetchone()
            self.conn.commit()
            if not current_quantity:
                print(f'No data found for InventoryID: {inventory_id}')
                return False

            current_quantity = current_quantity[0]

            if current_quantity >= quantity_to_check:
                print(f"Product with quantity {quantity_to_check} is available")
            else:
                self.close()
                print(f"Product with quantity {quantity_to_check} is not available")
                print(f"Product has a total of {current_quantity} units only ")
        except Exception as e:
            print(e)
            return False

    def GetInventoryValue(self):
        try:
            self.open()
            get_inventory_value_query = '''
                SELECT SUM(Products.Price * Inventory.QuantityInStock) AS TotalValue
                FROM Inventory
                INNER JOIN Products ON Inventory.ProductID = Products.Product_ID
            '''
            self.stmt.execute(get_inventory_value_query)
            records = self.stmt.fetchall()
            for i in records:
                print(f"Total Inventory Value={i[0]}")
            # else:
            #     print("data is not there")
            self.close()
        except Exception as e:
            print(e)

    def ListLowStockProducts(self):
        try:
            threshold = int(input("enter threshold"))
            if not isinstance(threshold, int) or threshold < 0:
                raise InvalidIDError()
            self.open()
            statement = '''SELECT p.Product_Name, i.QuantityInStock
                        FROM Products p
                        JOIN Inventory i ON p.Product_ID = i.ProductID
                        WHERE i.QuantityInStock < %s'''
            self.stmt.execute(statement, (threshold,))
            result = self.stmt.fetchall()
            if result:
                for i in result:
                    print(f"Low Stock Product: {i[0]}")
            else:
                print("all are more ")
            self.close()
        except Exception as e:
            print(e)

    def ListOutOfStockProducts(self):
        try:
            self.open()
            statement = '''SELECT p.Product_Name, i.QuantityInStock
```

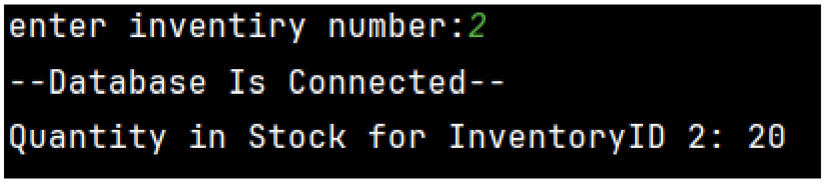
```
                        FROM Products p
                        JOIN Inventory i ON p.Product_ID = i.ProductID
                        WHERE i.QuantityInStock =0'''
            self.stmt.execute(statement)
            result = self.stmt.fetchall()
            if result:
                for i in result:
                    print(f"Product  {i[0]} is out of stock")
            else:
                print("all are available")

            self.close()
        except Exception as e:
            print(e)
```

GetProduct O/P:



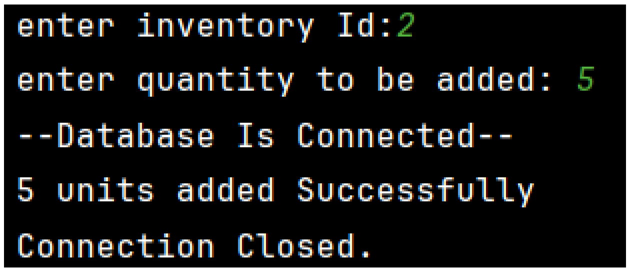
```
Enter Product ID to get product details: 2
--Database Is Connected--

Product Details:
ProductID: 2
ProductName: Camera
Description: Latest model with dual camera
Price: 769
```

GetQuantityInStock O/P:



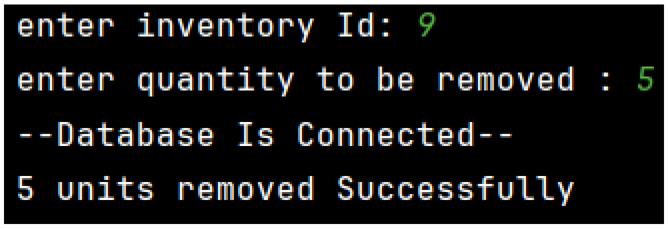
```
enter inventiry number:2
--Database Is Connected--
Quantity in Stock for InventoryID 2: 20
```

AddToInventory O/P:



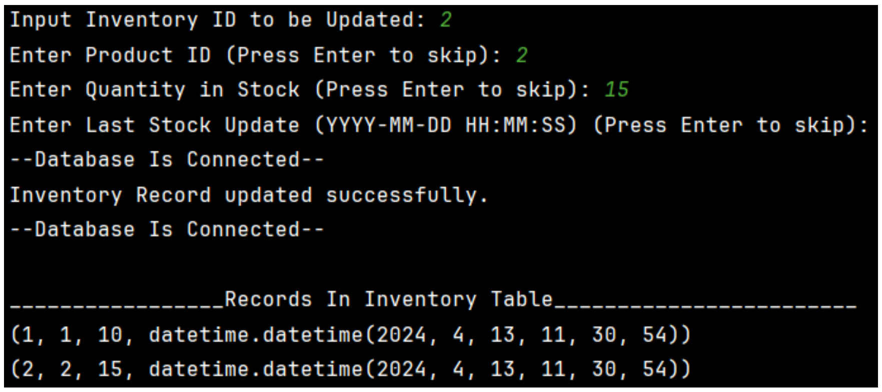
```
enter inventory Id:2
enter quantity to be added: 5
--Database Is Connected--
5 units added Successfully
Connection Closed.
```

RemoveFromInventory O/P:



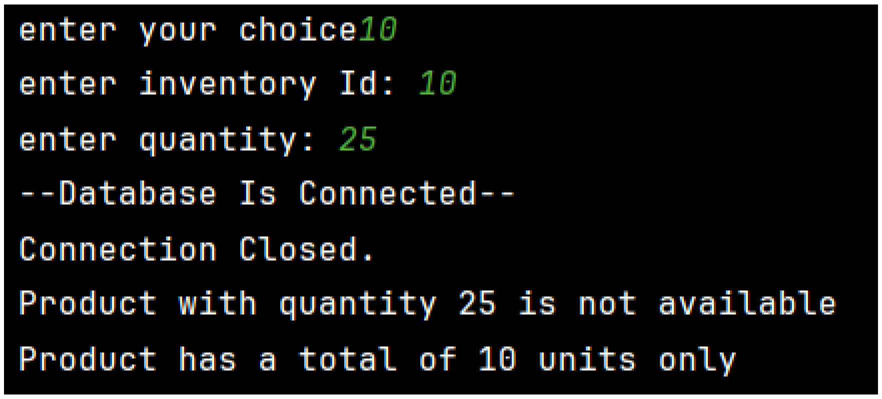
```
enter inventory Id: 9
enter quantity to be removed : 5
--Database Is Connected--
5 units removed Successfully
```

UpdateStockQuantity O/P:



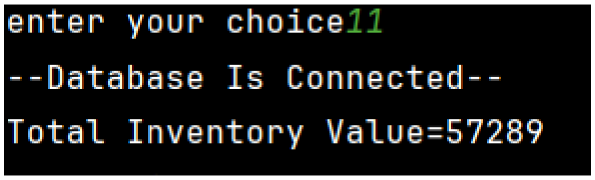
```
Input Inventory ID to be Updated: 2
Enter Product ID (Press Enter to skip): 2
Enter Quantity in Stock (Press Enter to skip): 15
Enter Last Stock Update (YYYY-MM-DD HH:MM:SS) (Press Enter to skip):
--Database Is Connected--
Inventory Record updated successfully.
--Database Is Connected--

_________________Records In Inventory Table_________________________
(1, 1, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
(2, 2, 15, datetime.datetime(2024, 4, 13, 11, 30, 54))
```

IsProductAvailable O/P:



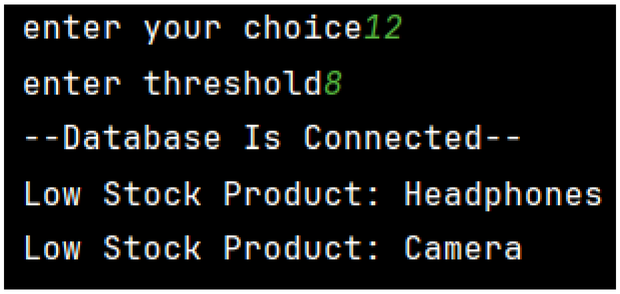
```
enter your choice10
enter inventory Id: 10
enter quantity: 25
--Database Is Connected--
Connection Closed.
Product with quantity 25 is not available
Product has a total of 10 units only
```

GetInventoryValue O/P:



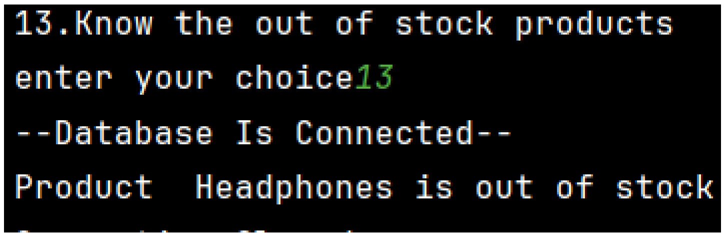
```
enter your choice11
--Database Is Connected--
Total Inventory Value=57289
```

ListLowStockProducts O/P:



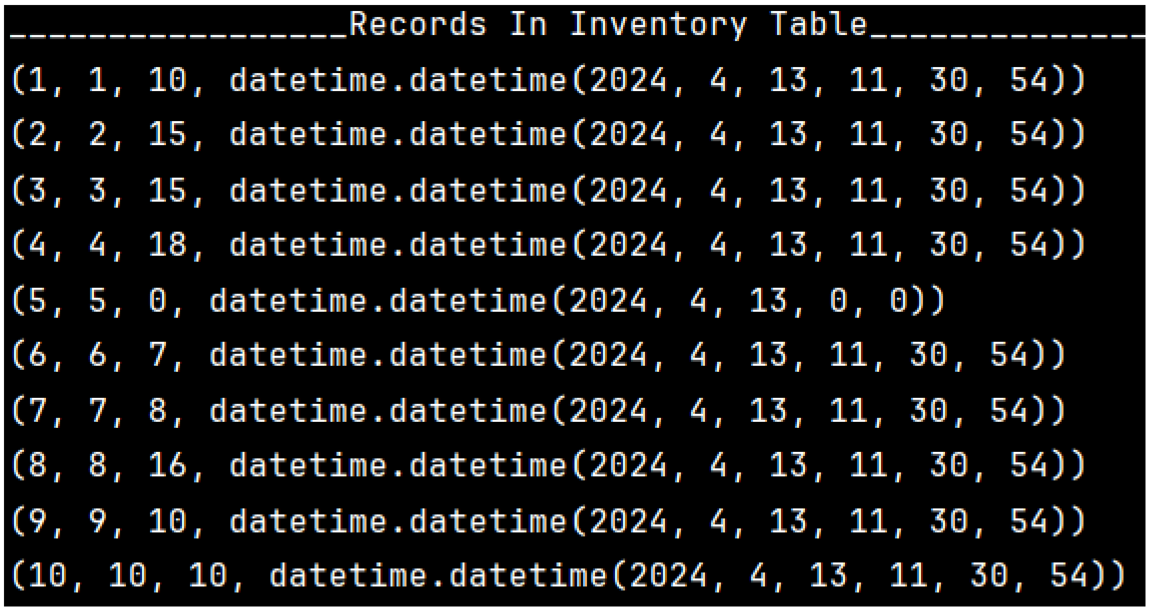
```
enter your choice12
enter threshold8
--Database Is Connected--
Low Stock Product: Headphones
Low Stock Product: Camera
```

ListOutOfStockProducts O/P:



```
13.Know the out of stock products
enter your choice13
--Database Is Connected--
Product  Headphones is out of stock
```

ListAllProducts O/P:



```
_________________Records In Inventory Table______________
(1, 1, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
(2, 2, 15, datetime.datetime(2024, 4, 13, 11, 30, 54))
(3, 3, 15, datetime.datetime(2024, 4, 13, 11, 30, 54))
(4, 4, 18, datetime.datetime(2024, 4, 13, 11, 30, 54))
(5, 5, 0, datetime.datetime(2024, 4, 13, 0, 0))
(6, 6, 7, datetime.datetime(2024, 4, 13, 11, 30, 54))
(7, 7, 8, datetime.datetime(2024, 4, 13, 11, 30, 54))
(8, 8, 16, datetime.datetime(2024, 4, 13, 11, 30, 54))
(9, 9, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
(10, 10, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
```

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.

CUSTOMERS CLASS:

```python
class Customers:
    def __init__(self, customer_id, first_name, last_name, email, phone, address):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.address = address

    def UpdateCustomerInfo(self, new_email, new_phone, new_address):
        self.email = new_email
        self.phone = new_phone
        self.address = new_address

    def CalculateTotalOrders(self):
        pass


    def GetCustomerDetails(self):
        details = (
            f"Customer Details: {self.first_name} {self.last_name}\n"
            f"Email: {self.email}\n"
            f"Phone: {self.phone}\n"
            f"Address: {self.address}"
        )
        return details
```

PRODUCTS CLASS:

```python
class Products:
    def __init__(self, product_id, product_name, description, price):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price

    def UpdateProductInfo(self, new_description, new_price):
        self.description = new_description
        self.price = new_price

    def IsProductInStock(self):
        pass

    def GetProductDetails(self):
        details = (
            f"Product Details: {self.product_name}\n"
            f"Product ID: {self.product_id}\n"
            f"Description: {self.description}\n"
            f"Price: ${self.price}"
        )
        return details
```

INVENTORY CLASS:

```python
from datetime import datetime
from Products import Products
class Inventory:
    def __init__(self, inventoryID, product, quantityInStock, lastStockUpdate):
        self.inventoryID = inventoryID
        self.product = product
        self.quantityInStock = quantityInStock
        self.lastStockUpdate = lastStockUpdate

    def GetProduct(self):
        return self.product

    def GetQuantityInStock(self):
        return self.quantityInStock

    def AddToInventory(self, quantity):
        pass

    def RemoveFromInventory(self, quantity):
        pass

    def UpdateStockQuantity(self, newQuantity):
        pass

    def IsProductAvailable(self, quantityToCheck):
        pass

    def GetInventoryValue(self):
        pass

    def ListLowStockProducts(self, threshold):
        pass

    def ListOutOfStockProducts(self):
        pass
```

ORDERDETAILS CLASS:

```python
class OrderDetails:
    def __init__(self, order_detail_id, order, product, quantity):
        self.order_detail_id = order_detail_id
        self.order = order
        self.product = product
        self.quantity = quantity

    def CalculateSubtotal(self):
        pass


    def GetOrderDetailInfo(self):
        info = (
            f"Order Detail ID: {self.order_detail_id}\n"
            f"Product: {self.product.product_name}\n"
            f"Quantity: {self.quantity}\n"
        )
        return info

    def UpdateQuantity(self, new_quantity):
        pass

    def AddDiscount(self, discount_amount):
        pass
```

ORDERS CLASS:

```python
from datetime import datetime
from Customers import Customers


class Orders:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.order_id = order_id
        self.customer = customer
        self.order_date = order_date
        self.total_amount = total_amount

    def CalculateTotalAmount(self):
        pass

    def GetOrderDetails(self):
        details = (
            f"Order ID: {self.order_id}\n"
            f"Order Date: {self.order_date}\n"
            f"Total Amount: ${self.total_amount}\n"
            f"Customer Details:\n{self.customer.GetCustomerDetails()}"
        )
        return details

    def UpdateOrderStatus(self, new_status):
        pass

    def CancelOrder(self):
        pass
```

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).

For Customers:

```python
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

    def set_customerId(self, customerId):
        self.customerId = customerId

    def set_firstName(self, firstName):
        self.firstName = firstName

    def set_lastName(self, lastName):
        self.lastName = lastName

    def set_email(self, email):
        self.email = email

    def set_phone(self, phone):
        self.phone = phone

    def set_address(self, address):
        self.address = address

    def get_customerId(self):
        return self.customerId

    def get_firstName(self):
        return self.firstName

    def get_lastName(self):
        return self.lastName

    def get_email(self):
        return self.email

    def get_phone(self):
        return self.phone

    def get_address(self):
        return self.address
```

For Orders:

```
class Order:
    def __init__(self, orderId, customer, orderDate, totalAmount):
        self.orderId = orderId
        self.customer = customer
        self.orderDate = orderDate
        self.totalAmount = totalAmount

    def set_orderId(self, orderId):
        self.orderId = orderId

    def set_customer(self, customer):
        self.customer = customer

    def set_orderDate(self, orderDate):
        self.orderDate = orderDate

    def set_totalAmount(self, totalAmount):
        self.totalAmount = totalAmount

    def get_orderId(self):
        return self.orderId

    def get_customer(self):
        return self.customer

    def get_orderDate(self):
        return self.orderDate

    def get_totalAmount(self):
        return self.totalAmount
```

For OrderDetails:

```
class OrderDetails:
    def __init__(self, orderDetailsId, order, product, quantity):
        self.orderDetailsId = orderDetailsId
        self.order = order
        self.product = product
        self.quantity = quantity

    def set_orderDetailsId(self, orderDetailsId):
        self.orderDetailsId = orderDetailsId

    def set_order(self, order):
        self.order = order

    def set_product(self, product):
        self.product = product

    def set_quantity(self, quantity):
        self.quantity = quantity

    def get_orderDetailsId(self):
        return self.orderDetailsId

    def get_order(self):
        return self.order

    def get_product(self):
        return self.product
    def get_quantity(self):
        return self.quantity
```

For Products:

```
class Product:
    def __init__(self, productId, productName, description, price):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price

    def set_productId(self, productId):
        self.productId = productId

    def set_productName(self, productName):
        self.productName = productName

    def set_description(self, description):
        self.description = description

    def set_price(self, price):
        self.price = price

    def get_productId(self):
        return self.productId

    def get_productName(self):
        return self.productName

    def get_description(self):
        return self.description

    def get_price(self):
        return self.price
```

For Inventory:

```
class Inventory:
    def __init__(self, inventoryId, product, quantityInStock, lastStockUpdate):
        self.inventoryId = inventoryId
        self.product = product
        self.quantityInStock = quantityInStock
        self.lastStockUpdate = lastStockUpdate

    def set_inventoryId(self, inventoryId):
        self.inventoryId = inventoryId

    def set_product(self, product):
        self.product = product

    def set_quantityInStock(self, quantityInStock):
        self.quantityInStock = quantityInStock

    def set_lastStockUpdate(self, lastStockUpdate):
        self.lastStockUpdate = lastStockUpdate

    def get_inventoryId(self):
        return self.inventoryId

    def get_product(self):
        return self.product
    def get_quantityInStock(self):
        return self.quantityInStock

    def get_lastStockUpdate(self):
        return self.lastStockUpdate
```

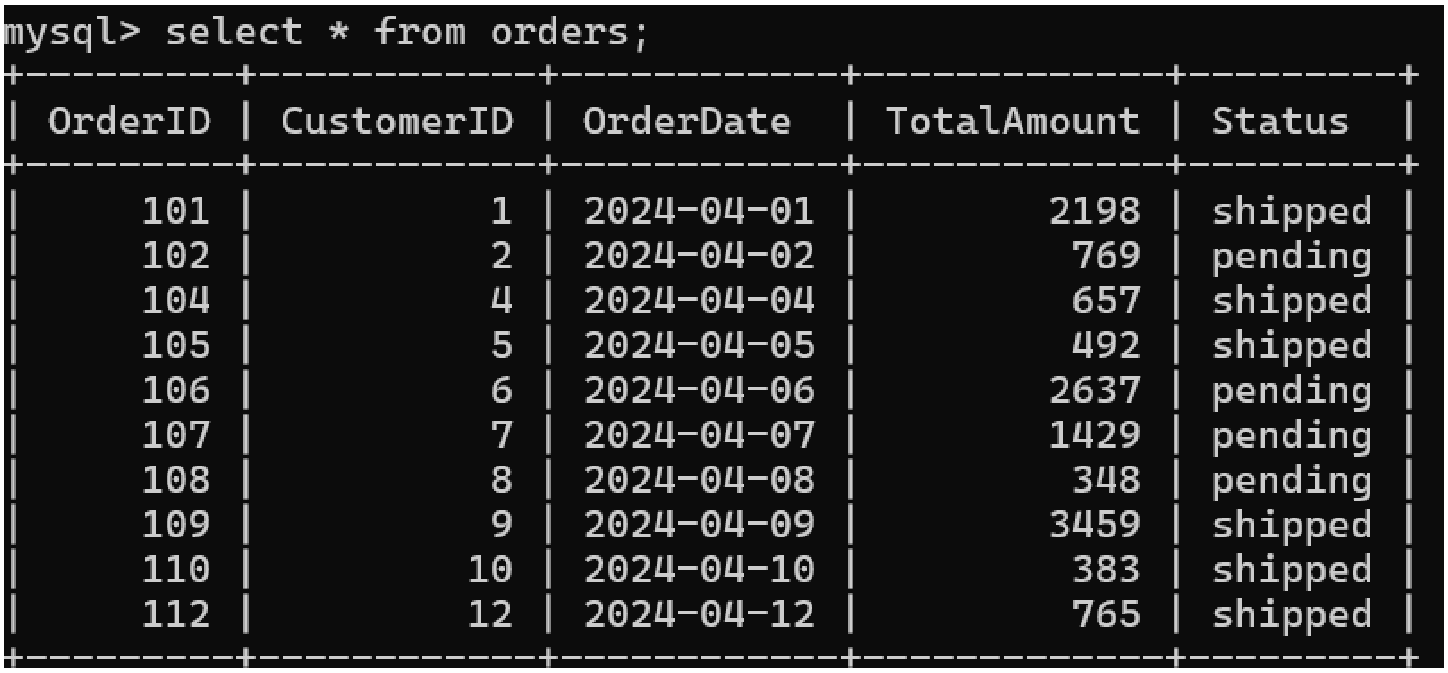
Task 4: Composition:

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

• **Orders Class with Composition:**

- In the Orders class, we want to establish a composition relationship with the Customers class, indicating that each order is associated with a specific customer.
- In the Orders class, we've added a private attribute customer of type Customers, establishing a composition relationship. The Customer property provides access to the Customers object associated with the order

```
CREATE TABLE IF NOT EXISTS Orders (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(7, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE
);
'''
```



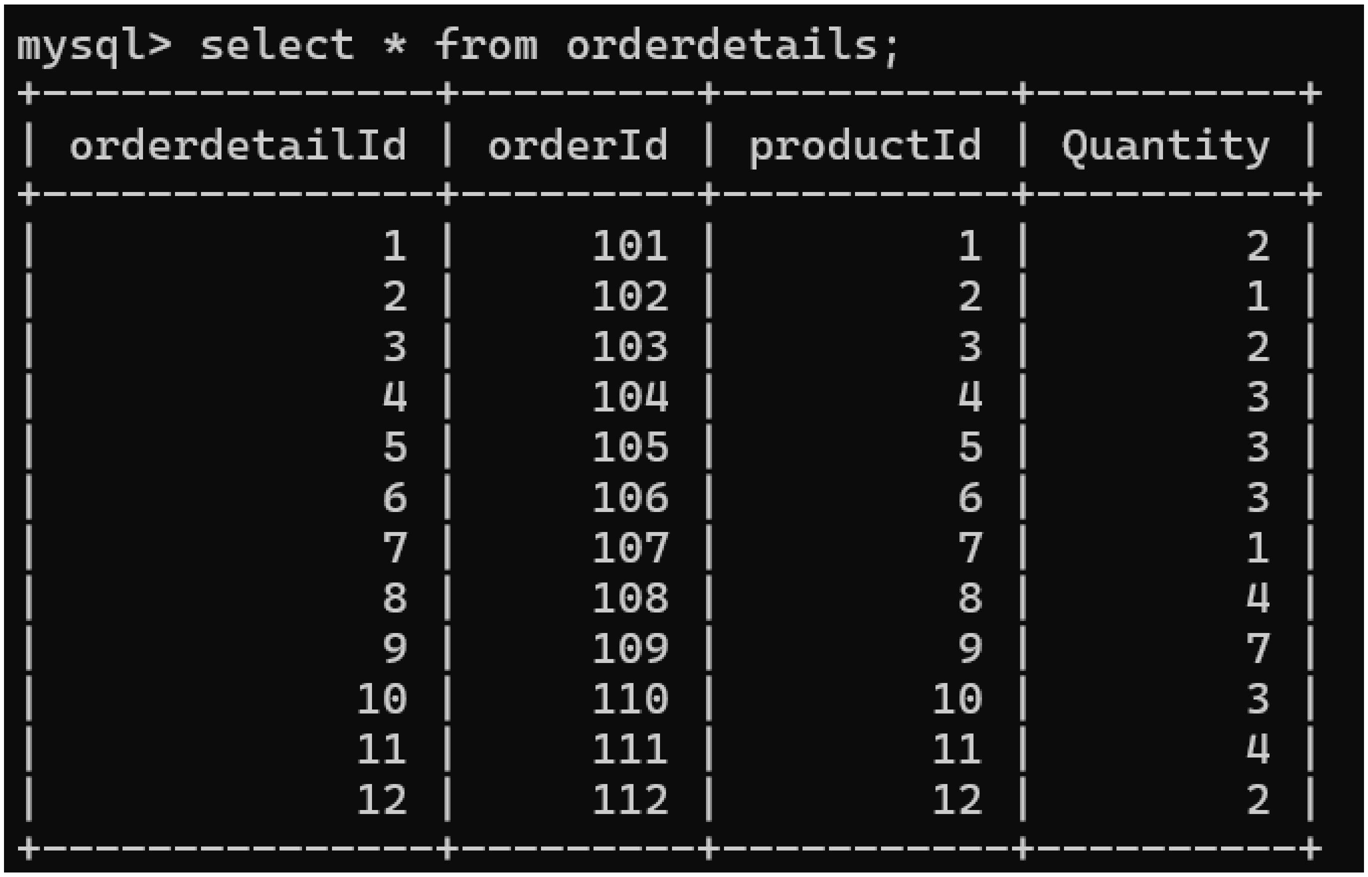
```
mysql> select * from orders;
+---------+------------+------------+-------------+---------+
| OrderID | CustomerID | OrderDate  | TotalAmount | Status  |
+---------+------------+------------+-------------+---------+
|     101 |          1 | 2024-04-01 |        2198 | shipped |
|     102 |          2 | 2024-04-02 |         769 | pending |
|     104 |          4 | 2024-04-04 |         657 | shipped |
|     105 |          5 | 2024-04-05 |         492 | shipped |
|     106 |          6 | 2024-04-06 |        2637 | pending |
|     107 |          7 | 2024-04-07 |        1429 | pending |
|     108 |          8 | 2024-04-08 |         348 | pending |
|     109 |          9 | 2024-04-09 |        3459 | shipped |
|     110 |         10 | 2024-04-10 |         383 | shipped |
|     112 |         12 | 2024-04-12 |         765 | shipped |
+---------+------------+------------+-------------+---------+
```

• **OrderDetails Class with Composition:**

- Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered.

- In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the

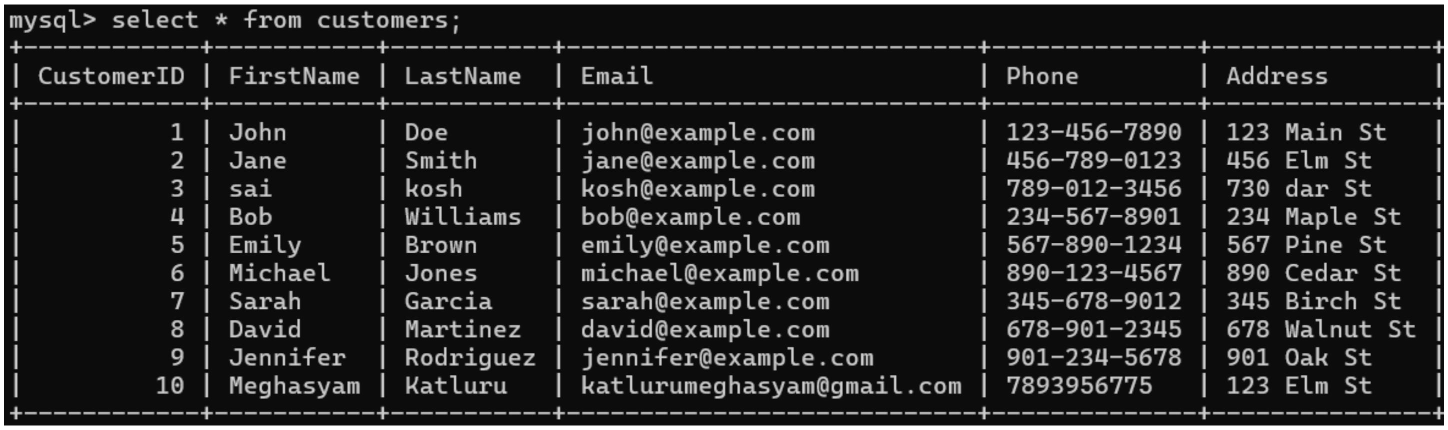
```
CREATE TABLE IF NOT EXISTS OrderDetails (
    OrderDetailID INT PRIMARY KEY AUTO_INCREMENT,
    OrderID INT,
    ProductID INT,
    Quantity INT,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE CASCADE
);
'''
```



```
mysql> select * from orderdetails;
+---------------+---------+-----------+----------+
| orderdetailId | orderId | productId | Quantity |
+---------------+---------+-----------+----------+
|             1 |     101 |         1 |        2 |
|             2 |     102 |         2 |        1 |
|             3 |     103 |         3 |        2 |
|             4 |     104 |         4 |        3 |
|             5 |     105 |         5 |        3 |
|             6 |     106 |         6 |        3 |
|             7 |     107 |         7 |        1 |
|             8 |     108 |         8 |        4 |
|             9 |     109 |         9 |        7 |
|            10 |     110 |        10 |        3 |
|            11 |     111 |        11 |        4 |
|            12 |     112 |        12 |        2 |
+---------------+---------+-----------+----------+
```

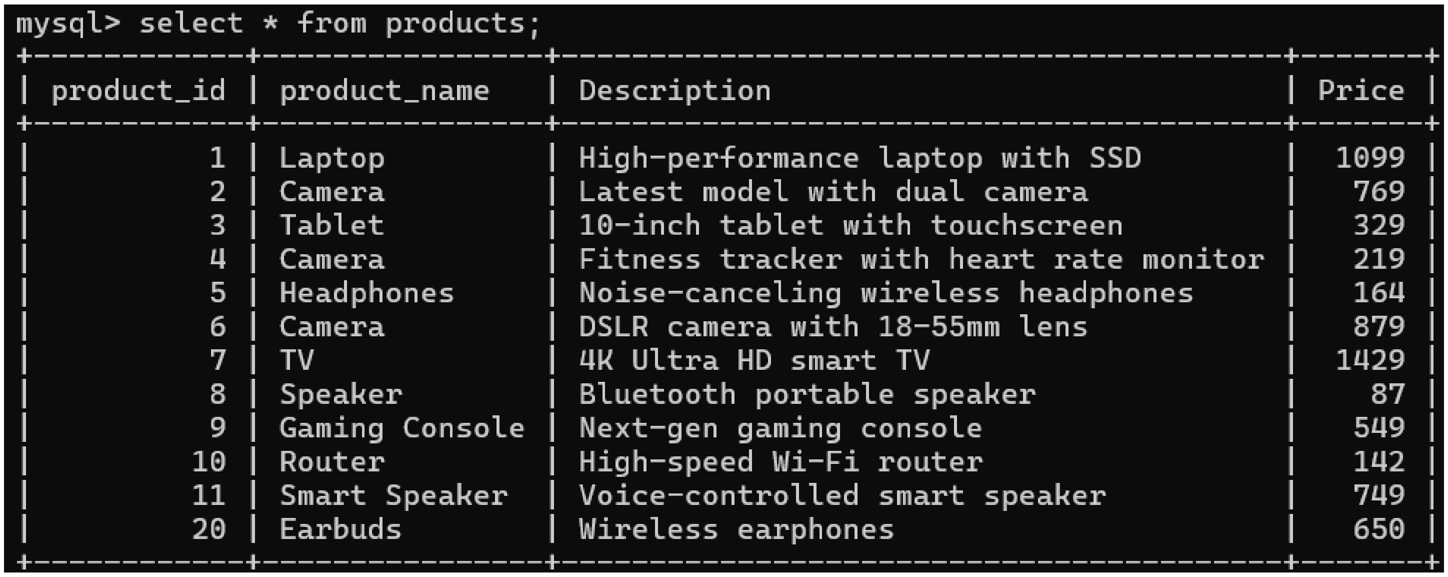
- **Customers and Products Classes:**
  - The Customers and Products classes themselves may not have direct composition relationships with other classes in this scenario. However, they serve as the basis for

```
composition relationships in the Orders and OrderDetails classes, respectively.
 '''
CREATE TABLE IF NOT EXISTS Customers (
CustomerID INT PRIMARY KEY AUTO_INCREMENT,
FirstName VARCHAR(55),
LastName VARCHAR(55),
Email VARCHAR(55),
Phone VARCHAR(20),
Address VARCHAR(55)
        );
'''
```



```
mysql> select * from customers;
+------------+-----------+-----------+----------------------------+--------------+----------------+
| CustomerID | FirstName | LastName  | Email                      | Phone        | Address        |
+------------+-----------+-----------+----------------------------+--------------+----------------+
|          1 | John      | Doe       | john@example.com           | 123-456-7890 | 123 Main St    |
|          2 | Jane      | Smith     | jane@example.com           | 456-789-0123 | 456 Elm St     |
|          3 | sai       | kosh      | kosh@example.com           | 789-012-3456 | 730 dar St     |
|          4 | Bob       | Williams  | bob@example.com            | 234-567-8901 | 234 Maple St   |
|          5 | Emily     | Brown     | emily@example.com          | 567-890-1234 | 567 Pine St    |
|          6 | Michael   | Jones     | michael@example.com        | 890-123-4567 | 890 Cedar St   |
|          7 | Sarah     | Garcia    | sarah@example.com          | 345-678-9012 | 345 Birch St   |
|          8 | David     | Martinez  | david@example.com          | 678-901-2345 | 678 Walnut St  |
|          9 | Jennifer  | Rodriguez | jennifer@example.com       | 901-234-5678 | 901 Oak St     |
|         10 | Meghasyam | Katluru   | katlurumeghasyam@gmail.com | 7893956775   | 123 Elm St     |
+------------+-----------+-----------+----------------------------+--------------+----------------+
```

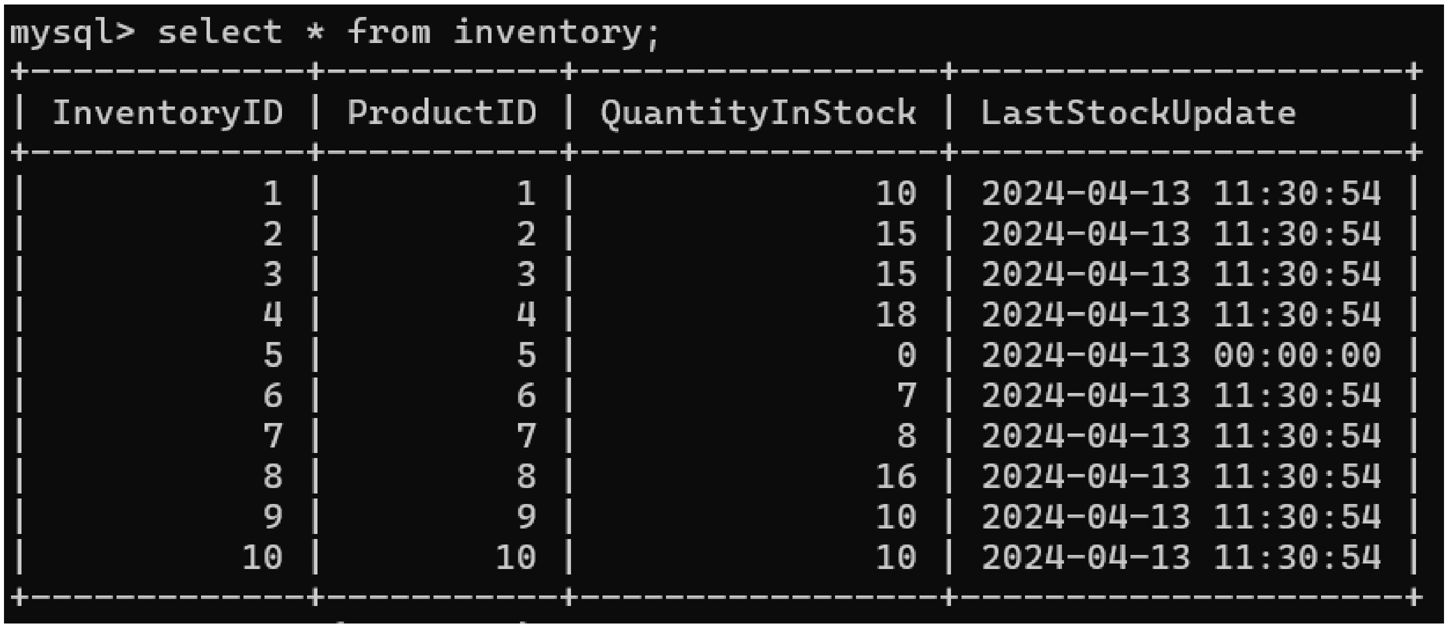
```
'''
CREATE TABLE IF NOT EXISTS Products (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    ProductName VARCHAR(55),
    Description VARCHAR(55),
    Price INT
);
'''
```



```
mysql> select * from products;
+------------+----------------+------------------------------------------+-------+
| product_id | product_name   | Description                              | Price |
+------------+----------------+------------------------------------------+-------+
|          1 | Laptop         | High-performance laptop with SSD         |  1099 |
|          2 | Camera         | Latest model with dual camera            |   769 |
|          3 | Tablet         | 10-inch tablet with touchscreen          |   329 |
|          4 | Camera         | Fitness tracker with heart rate monitor  |   219 |
|          5 | Headphones     | Noise-canceling wireless headphones      |   164 |
|          6 | Camera         | DSLR camera with 18-55mm lens            |   879 |
|          7 | TV             | 4K Ultra HD smart TV                     |  1429 |
|          8 | Speaker        | Bluetooth portable speaker               |    87 |
|          9 | Gaming Console | Next-gen gaming console                  |   549 |
|         10 | Router         | High-speed Wi-Fi router                  |   142 |
|         11 | Smart Speaker  | Voice-controlled smart speaker           |   749 |
|         20 | Earbuds        | Wireless earphones                       |   650 |
+------------+----------------+------------------------------------------+-------+
```

- **Inventory Class:**
    - The Inventory class represents the inventory of products available for sale. It can have composition relationships with the Products class to indicate which products are in the inventory

```
'''
CREATE TABLE IF NOT EXISTS Inventory (
    InventoryID INT PRIMARY KEY AUTO_INCREMENT,
    ProductID INT,
    QuantityInStock INT,
    LastStockUpdate DATE,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE CASCADE
);
```



```
mysql> select * from inventory;
+-------------+-----------+-----------------+---------------------+
| InventoryID | ProductID | QuantityInStock | LastStockUpdate     |
+-------------+-----------+-----------------+---------------------+
|           1 |         1 |              10 | 2024-04-13 11:30:54 |
|           2 |         2 |              15 | 2024-04-13 11:30:54 |
|           3 |         3 |              15 | 2024-04-13 11:30:54 |
|           4 |         4 |              18 | 2024-04-13 11:30:54 |
|           5 |         5 |               0 | 2024-04-13 00:00:00 |
|           6 |         6 |               7 | 2024-04-13 11:30:54 |
|           7 |         7 |               8 | 2024-04-13 11:30:54 |
|           8 |         8 |              16 | 2024-04-13 11:30:54 |
|           9 |         9 |              10 | 2024-04-13 11:30:54 |
|          10 |        10 |              10 | 2024-04-13 11:30:54 |
+-------------+-----------+-----------------+---------------------+
```

Task 5: Exceptions handling

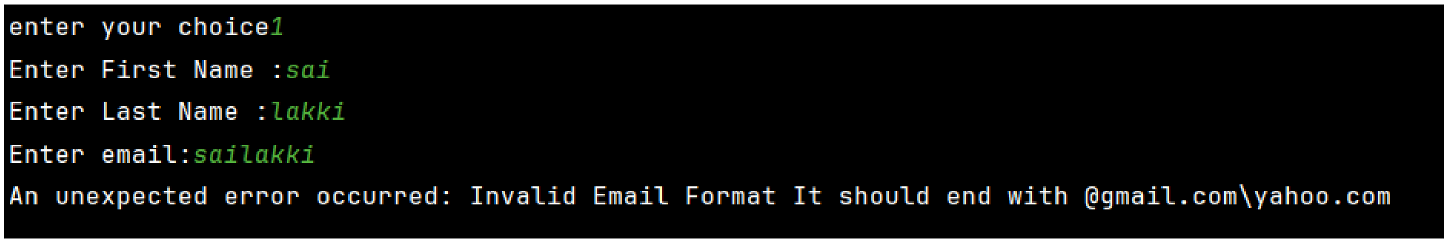
• Data Validation:

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

```
import re

class InvalidEmailError(Exception):
    def __init__(self, email, message="An unexpected error occurred: Invalid Email Format "
                                      "It should end with @gmail.com\yahoo.com"):
        self.email = email
        self.message = message
        super().__init__(self.message)




def validate_email(email):
    email_pattern = re.compile(r'^[a-zA-Z0-9._%+-]+@(gmail\.com|yahoo\.com)$')
    if not email_pattern.match(email):
        raise InvalidEmailError(email)
```



```
enter your choice1
Enter First Name :sai
Enter Last Name :lakki
Enter email:sailakki
An unexpected error occurred: Invalid Email Format It should end with @gmail.com\yahoo.com
```

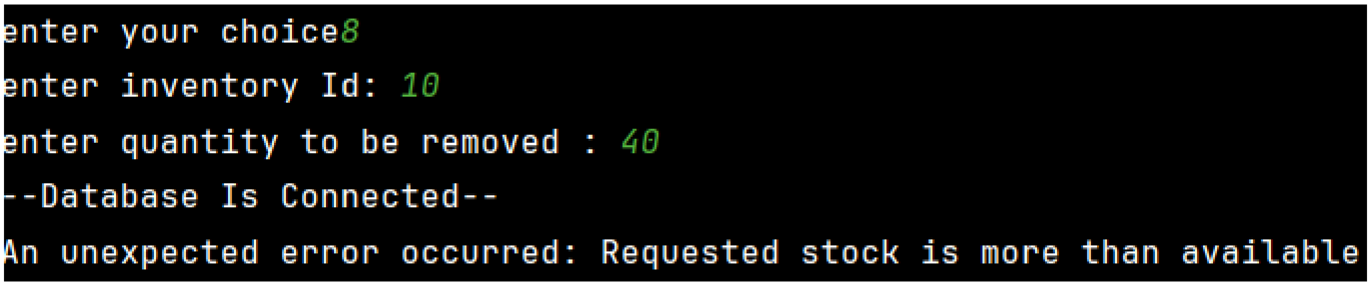
• **Inventory Management:**

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.

```
class InsufficientStockException(Exception):
    def __init__(self, message="Requested stock is more than available"):
        self.message = message
        super().__init__(self.message)
```



```
enter your choice8
enter inventory Id: 10
enter quantity to be removed : 40
--Database Is Connected--
An unexpected error occurred: Requested stock is more than available
```

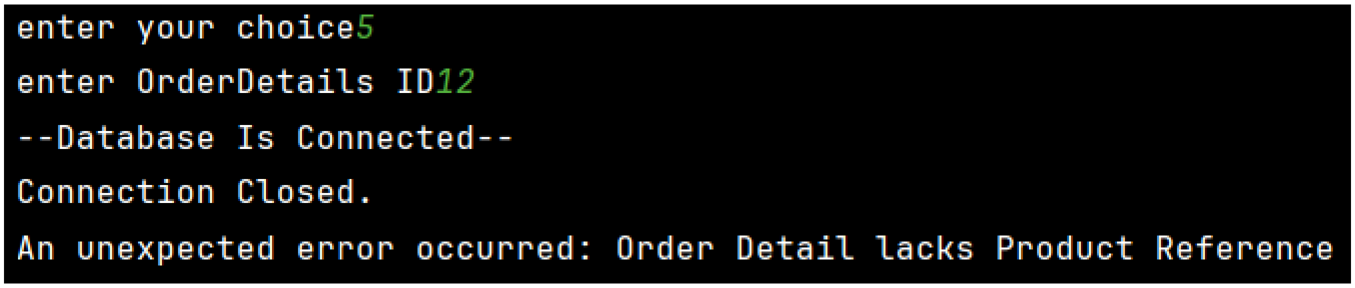
• **Order Processing:**

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.

```
class IncompleteOrderException(Exception):
    def __init__(self, message="Order Detail lacks Product Reference"):
        self.message = message
        super().__init__(self.message)
```



```
enter your choice5
enter OrderDetails ID12
--Database Is Connected--
Connection Closed.
An unexpected error occurred: Order Detail lacks Product Reference
```

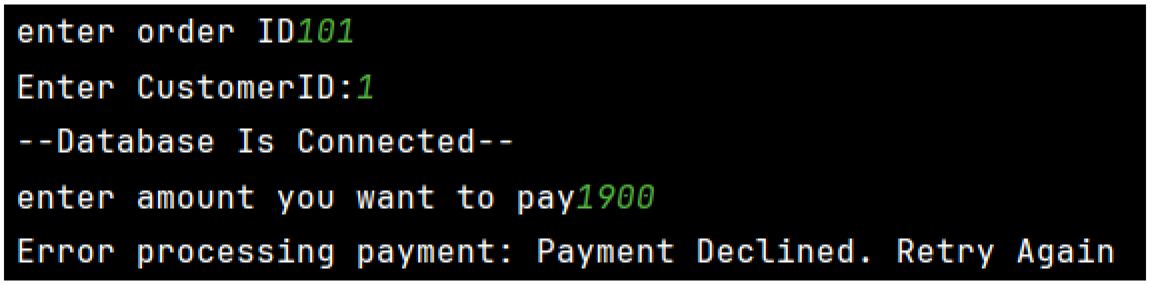
• **Payment Processing:**

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)

```
class PaymentFailedException(Exception):
    def __init__(self, message="Payment Declined. Retry Again"):
        self.message = message
        super().__init__(self.message)
```



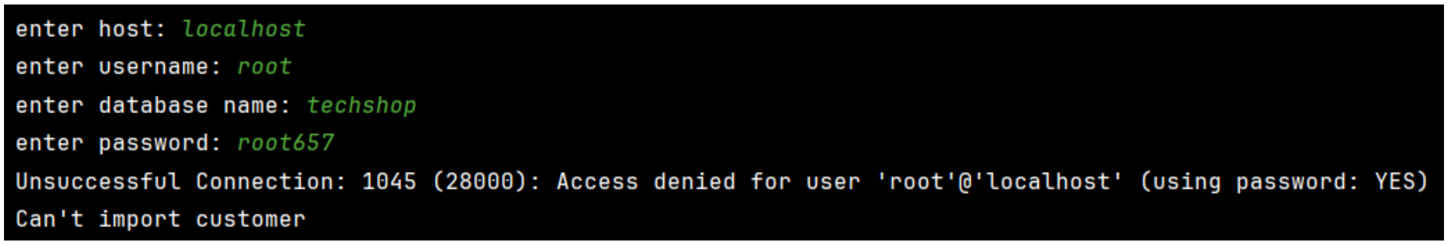
```
enter order ID101
Enter CustomerID:1
--Database Is Connected--
enter amount you want to pay1900
Error processing payment: Payment Declined. Retry Again
```

• **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.



```
enter host: localhost
enter username: root
enter database name: techshop
enter password: root657
Unsuccessful Connection: 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
Can't import customer
```

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.

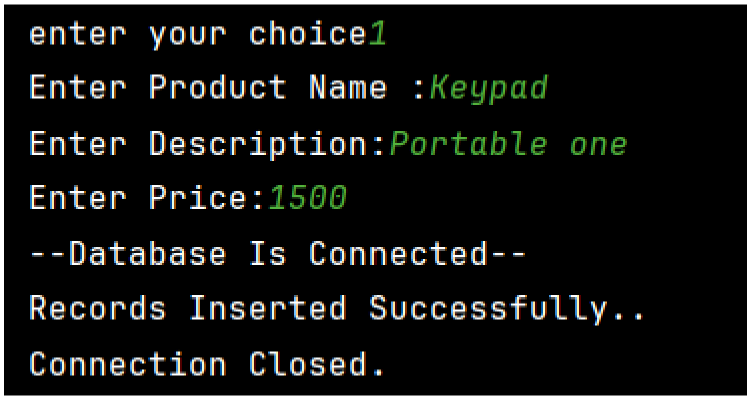
o Solution: Implement methods to add, update, and remove products. Handle exceptions for duplicate products

, invalid updates, or removal of products with existing orders.

Add Product:

```
class Products(dbConnection):
    def addProduct(self):
        Product_Name = input('Enter Product Name :')
        if not isinstance(Product_Name, str):
            raise InvalidNameError()
        StringCheck(Product_Name)
        self.Product_Name = Product_Name

        Description = input('Enter Description:')
        self.Description = Description

        new = int(input('Enter Price:'))
        if not isinstance(new, int) or new<0:
            raise InvalidPriceError()
        self.price = new

        data = [(self.Product_Name, self.Description, self.price)]
        insert_str = '''INSERT INTO Products (Product_Name, Description, Price)
                        VALUES (%s, %s, %s)'''

        self.open()
        self.stmt.executemany(insert_str, data)
        self.conn.commit()
        print('Records Inserted Successfully..')
        self.close()
```

O/P:

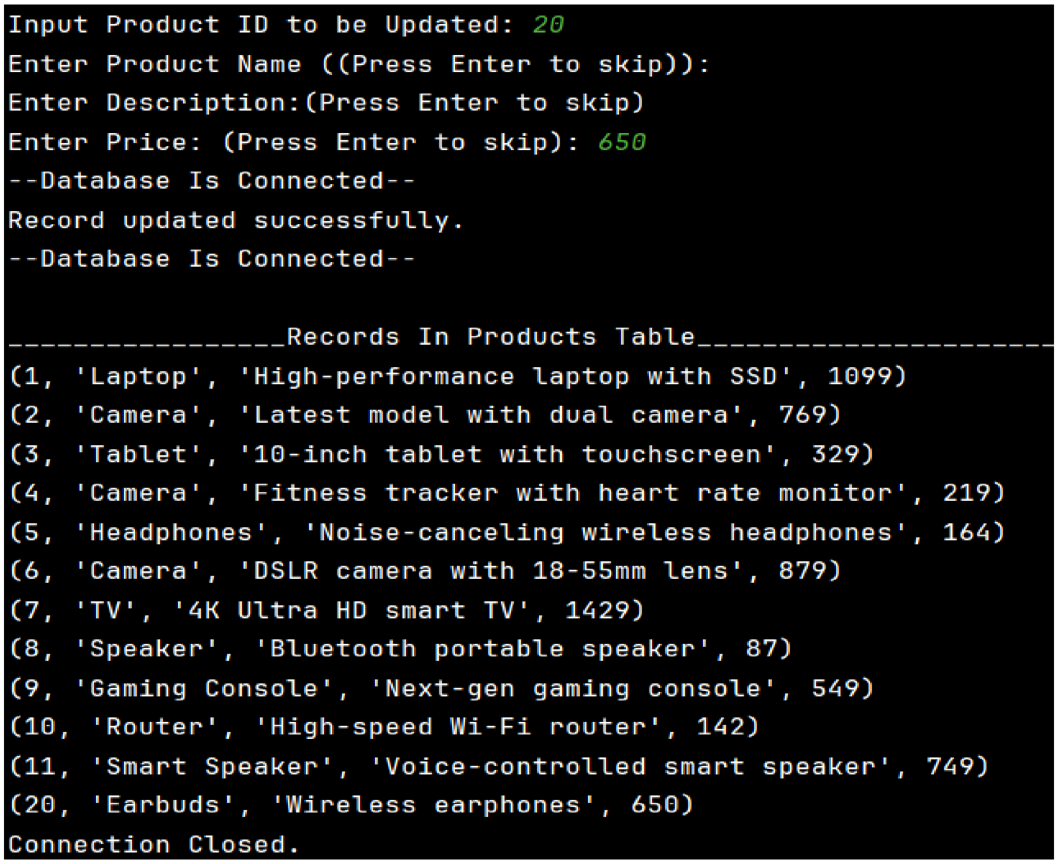


```
enter your choice1
Enter Product Name :Keypad
Enter Description:Portable one
Enter Price:1500
--Database Is Connected--
Records Inserted Successfully..
Connection Closed.
```

UpdateProducts :

```
def UpdateProductInfo(self):
    self.selectProducts()
    Product_id = int(input('Input Product ID to be Updated: '))
    if not isinstance(Product_id, int) or Product_id<0:
        raise InvalidIDError()
    Id = Product_id
    update_str = 'UPDATE Products SET '
    data = []

    Product_name = input('Enter Product Name ((Press Enter to skip)):')
    if Product_name:
        if not isinstance(Product_name, str):
            raise InvalidNameError()
        StringCheck(Product_name)
        self.ProductName = Product_name
        update_str += 'ProductName=%s, '
        data.append(self.ProductName)

    Description = input('Enter Description:(Press Enter to skip)')
    if Description:
        self.Description = Description
        update_str += 'Description=%s, '
        data.append(self.Description)

    Price = input('Enter Price: (Press Enter to skip): ')
    if Price:
        Price = int(Price)
        if not isinstance(Price, (int, float)) or Price < 0:
            raise InvalidPriceError()
        self.Price = int(Price)
        update_str += 'Price=%s, '
        data.append(self.Price)

    update_str = update_str.rstrip(', ')

    update_str += ' WHERE Product_ID=%s'
    data.append(Id)

    self.open()
    self.stmt.execute(update_str, data)
    self.conn.commit()
    print('Record updated successfully.')
    self.selectProducts()
```

O/P:

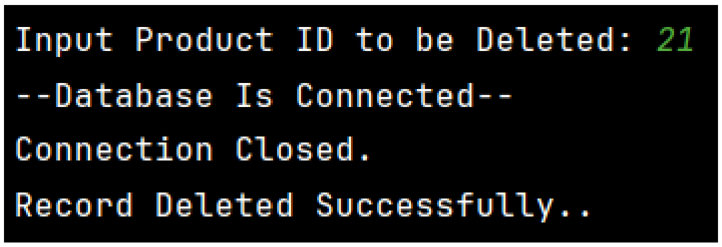


```
Input Product ID to be Updated: 20
Enter Product Name ((Press Enter to skip)):
Enter Description:(Press Enter to skip)
Enter Price: (Press Enter to skip): 650
--Database Is Connected--
Record updated successfully.
--Database Is Connected--

_________________Records In Products Table______________________
(1, 'Laptop', 'High-performance laptop with SSD', 1099)
(2, 'Camera', 'Latest model with dual camera', 769)
(3, 'Tablet', '10-inch tablet with touchscreen', 329)
(4, 'Camera', 'Fitness tracker with heart rate monitor', 219)
(5, 'Headphones', 'Noise-canceling wireless headphones', 164)
(6, 'Camera', 'DSLR camera with 18-55mm lens', 879)
(7, 'TV', '4K Ultra HD smart TV', 1429)
(8, 'Speaker', 'Bluetooth portable speaker', 87)
(9, 'Gaming Console', 'Next-gen gaming console', 549)
(10, 'Router', 'High-speed Wi-Fi router', 142)
(11, 'Smart Speaker', 'Voice-controlled smart speaker', 749)
(20, 'Earbuds', 'Wireless earphones', 650)
Connection Closed.
```

Remove Product:

```
def deleteProduct(self):
    self.selectProducts()
    Product_id = int(input('Input Product ID to be Deleted: '))
    if not isinstance(Product_id, int):
        raise InvalidIDError()
    else:
        if Product_id < 0:
            raise InvalidIDError()
    Id = Product_id
    delete_str = f'DELETE FROM Products WHERE Product_ID={Id}'
    self.open()
    self.stmt.execute(delete_str)
    self.conn.commit()
    self.close()
    print('Record Deleted Successfully..')
```

O/P:



```
Input Product ID to be Deleted: 21
--Database Is Connected--
Connection Closed.
Record Deleted Successfully..
```

• **Managing Orders List:**

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

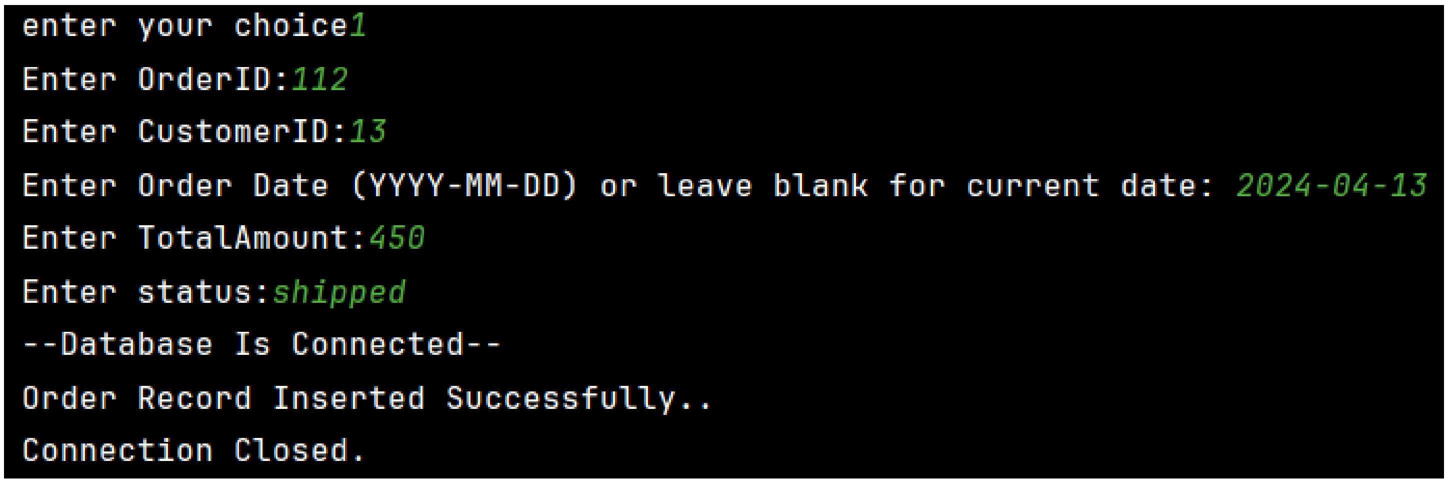
o Scenario: Adding new orders, updating order statuses, and removing canceled orders.

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.

Add Order:

```
class Orders(dbConnection):
    def addOrder(self):
        Order_ID = int(input('Enter OrderID:'))
        if not isinstance(Order_ID, int) or Order_ID < 0:
            raise InvalidIDError()
        self.Order_Id = Order_ID

        CustomerID = int(input('Enter CustomerID:'))
        if not isinstance(CustomerID, int) or CustomerID < 0:
            raise InvalidIDError()
        self.CustomerId = CustomerID
        order_date_input = input('Enter Order Date (YYYY-MM-DD) or leave blank for
current date: ')

        TotalAmount = int(input('Enter TotalAmount:'))
        if not isinstance(TotalAmount, (int, float)) or TotalAmount<0:
            raise InvalidPriceError()
        self.TotalAmount = TotalAmount

        Status = input('Enter status:')
        if not isinstance(Status, str):
            raise InvalidIDError()
        self.Status = Status

        if not order_date_input:
            order_date = datetime.now().strftime('%Y-%m-%d')
        else:
            order_date = order_date_input

        data = [(self.Order_Id, self.CustomerId, order_date, self.TotalAmount,
self.Status)]
        insert_order_str = '''INSERT INTO Orders (OrderId,CustomerID, OrderDate,
TotalAmount,Status)
                              VALUES (%s,%s,%s,%s, %s)'''

        self.open()
        self.stmt.executemany(insert_order_str, data)
        self.conn.commit()
        print('Order Record Inserted Successfully..')
        self.close()
```

O/P;

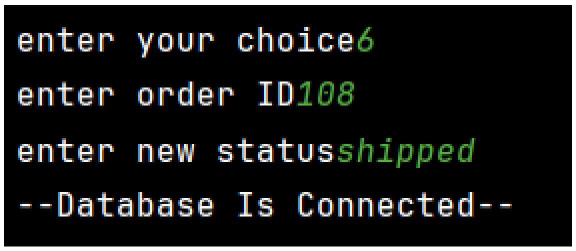


```
enter your choice1
Enter OrderID:112
Enter CustomerID:13
Enter Order Date (YYYY-MM-DD) or leave blank for current date: 2024-04-13
Enter TotalAmount:450
Enter status:shipped
--Database Is Connected--
Order Record Inserted Successfully..
Connection Closed.
```

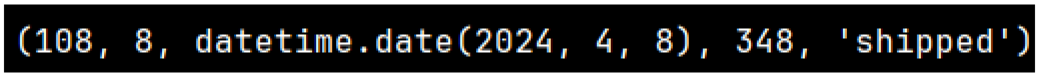
UpdateOrders:

```python
def updateOrderDetail(self):
    self.selectOrderDetails()
    order = int(input('Input OrderDetail ID to be Updated: '))
    if not isinstance(order, int) or order < 0:
        raise InvalidIDError()
    orderDetailId = order
    update_orderdetail_str = 'UPDATE OrderDetails SET '
    data = []

    self.orderID = int(input('Enter Order ID (Press Enter to skip): '))
    if self.orderID:

        update_orderdetail_str += 'OrderID=%s, '
        data.append(self.orderID)

    self.productID = int(input('Enter Product ID (Press Enter to skip): '))
    if self.productID:

        update_orderdetail_str += 'ProductID=%s, '
        data.append(self.productID)

    self.quantity = int(input('Enter Quantity (Press Enter to skip): '))
    if self.quantity:

        update_orderdetail_str += 'Quantity=%s, '
        data.append(self.quantity)

    update_orderdetail_str = update_orderdetail_str.rstrip(', ')

    update_orderdetail_str += ' WHERE OrderDetailID=%s'
    data.append(orderDetailId)

    self.open()
    self.stmt.execute(update_orderdetail_str, data)
    self.conn.commit()
    print('OrderDetail Record updated successfully.')
    self.selectOrderDetails()
```

O/P:



```
enter your choice6
enter order ID108
enter new statusshipped
--Database Is Connected--
```

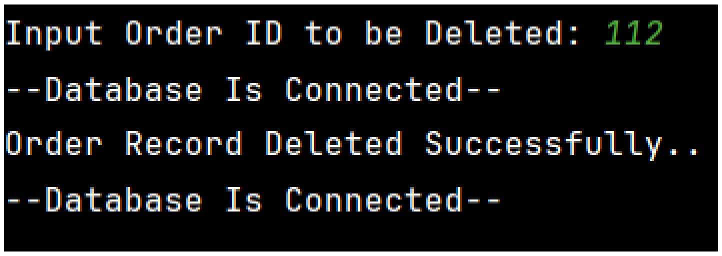


```
(108, 8, datetime.date(2024, 4, 8), 348, 'shipped')
```

Delete Order:

```
def deleteOrderDetail(self):
    OrderDetail_id = int(input('Input OrderDetail ID to be Deleted: '))
    if not isinstance(OrderDetail_id, int):
        raise InvalidIDError()
    else:
        if OrderDetail_id < 0:
            raise InvalidIDError()
    Id = OrderDetail_id
    delete_orderdetail_str = f'DELETE FROM OrderDetails WHERE OrderDetailID={Id}'
    self.open()
    self.stmt.execute(delete_orderdetail_str)
    self.conn.commit()
    print('OrderDetail Record Deleted Successfully..')
```

O/P



```
Input Order ID to be Deleted: 112
--Database Is Connected--
Order Record Deleted Successfully..
--Database Is Connected--
```

• **Sorting Orders by Date:**

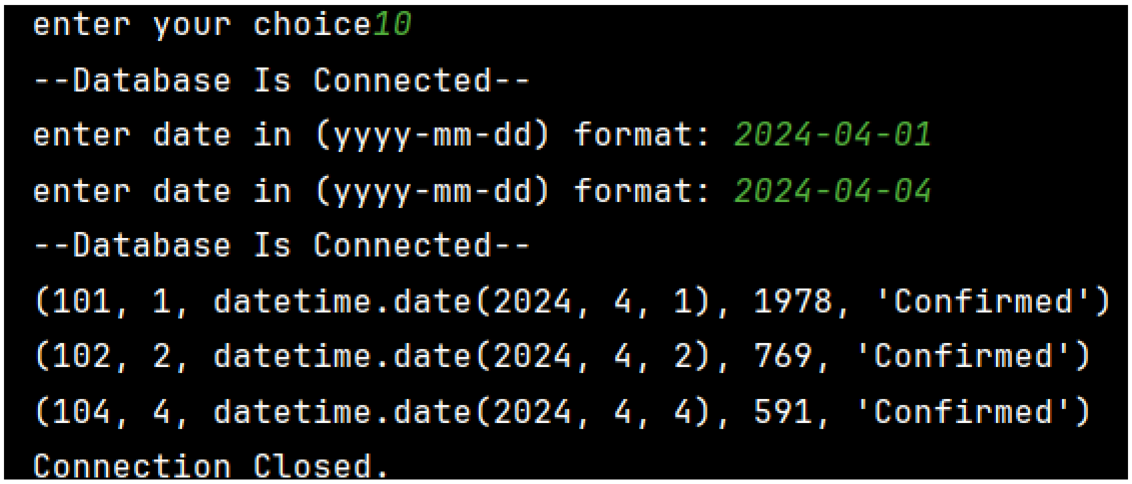
o Challenge: Sorting orders by order date in ascending or descending order.

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.

```
def selectOrdersWithinDate(self):
    self.open()
    startdate=input("enter date in (yyyy-mm-dd) format: ")
    enddate = input("enter date in (yyyy-mm-dd) format: ")
    select_orders_str = """SELECT * FROM Orders WHERE OrderDate BETWEEN %s AND %s"""
    self.open()
    data=(startdate,enddate)
    self.stmt.execute(select_orders_str,data)
    records = self.stmt.fetchall()

    for record in records:
        print(record)
    self.close()
```

O/P:



```
enter your choice10
--Database Is Connected--
enter date in (yyyy-mm-dd) format: 2024-04-01
enter date in (yyyy-mm-dd) format: 2024-04-04
--Database Is Connected--
(101, 1, datetime.date(2024, 4, 1), 1978, 'Confirmed')
(102, 2, datetime.date(2024, 4, 2), 769, 'Confirmed')
(104, 4, datetime.date(2024, 4, 4), 591, 'Confirmed')
Connection Closed.
```

• **Inventory Management with SortedList:**

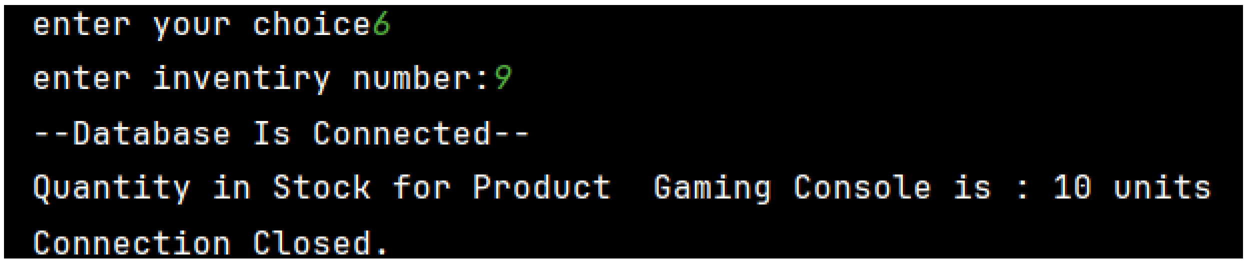
o Challenge: Managing product inventory with a SortedList based on product IDs.

o Scenario: Tracking the quantity in stock for each product and quickly retrieving inventory information.

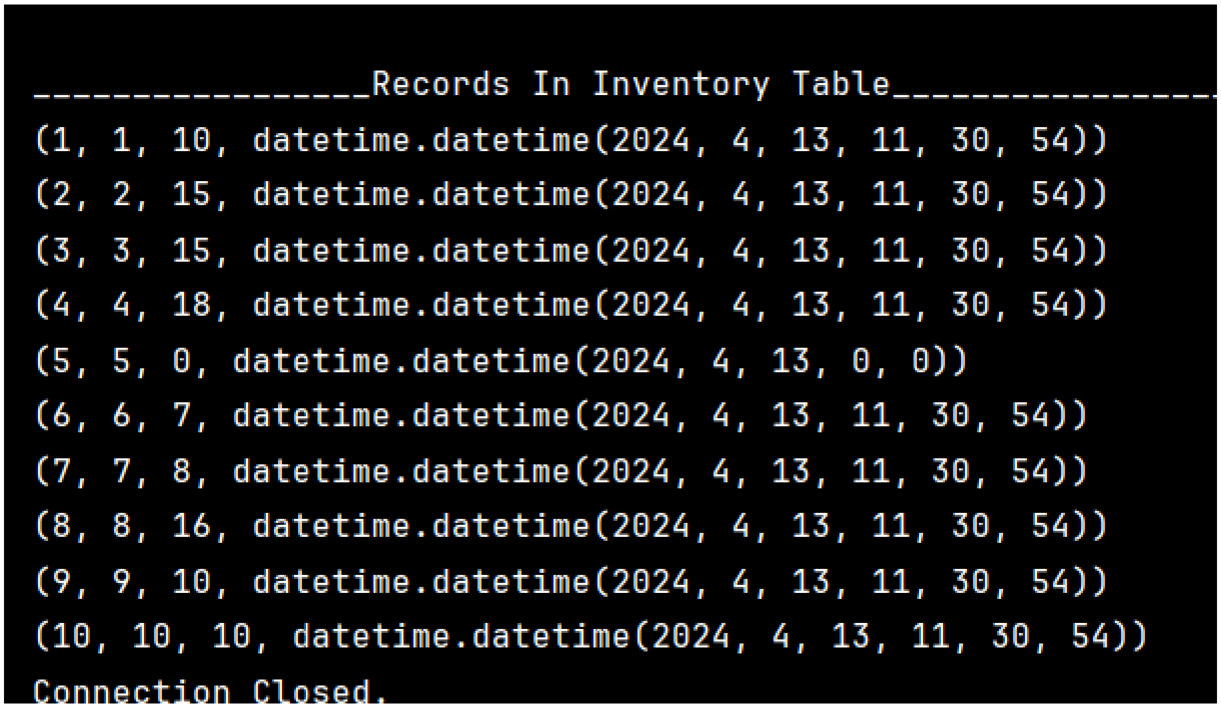
o Solution: Implement a SortedList<int, Inventory> where keys are product IDs. Ensure that inventory updates are synchronized with product additions and removals.

```
def GetQuantityInStock(self):
    try:
        inventory_id = int(input("enter inventiry number:"))
        if not isinstance(inventory_id, int) or inventory_id < 0:
            raise InvalidIDError()

        self.open()
        query = '''
            SELECT p.Product_Name, i.QuantityInStock
            FROM Products p
            JOIN Inventory i ON p.Product_ID = i.ProductID
            WHERE i.InventoryID = %s;

        '''

        self.stmt.execute(query, (inventory_id,))
        quantity_in_stock = self.stmt.fetchone()
        # print(quantity_in_stock)
        if quantity_in_stock:
            print(f'Quantity in Stock for Product  {quantity_in_stock[0]} is :
{quantity_in_stock[1]} units')
        else:
            print(f'No data found for InventoryID: {inventory_id}')
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```

O/P:



```
enter your choice6
enter inventiry number:9
--Database Is Connected--
Quantity in Stock for Product  Gaming Console is : 10 units
Connection Closed.
```



```
_________________Records In Inventory Table________________
(1, 1, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
(2, 2, 15, datetime.datetime(2024, 4, 13, 11, 30, 54))
(3, 3, 15, datetime.datetime(2024, 4, 13, 11, 30, 54))
(4, 4, 18, datetime.datetime(2024, 4, 13, 11, 30, 54))
(5, 5, 0, datetime.datetime(2024, 4, 13, 0, 0))
(6, 6, 7, datetime.datetime(2024, 4, 13, 11, 30, 54))
(7, 7, 8, datetime.datetime(2024, 4, 13, 11, 30, 54))
(8, 8, 16, datetime.datetime(2024, 4, 13, 11, 30, 54))
(9, 9, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
(10, 10, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
Connection Closed.
```

• **Handling Inventory Updates:**

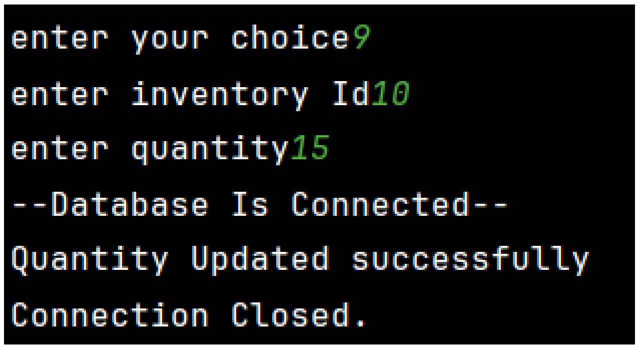
o Challenge: Ensuring that inventory is updated correctly when processing orders.

o Scenario: Decrementing product quantities in stock when orders are placed.

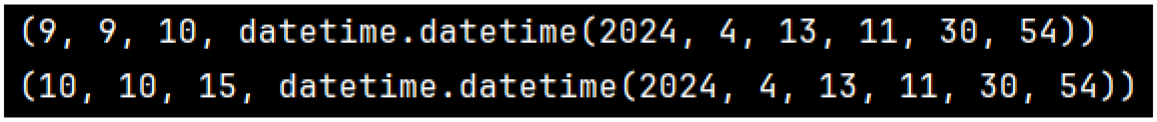
o Solution: Implement a method to update inventory quantities when orders are processed. Handle exceptions for insufficient stock

```
def UpdateStockInventory(self):
    try:
        inventory_id = int(input("enter inventory Id"))
        if not isinstance(inventory_id, int) or inventory_id < 0:
            raise InvalidIDError()
        quantity = int(input("enter quantity"))
        if not isinstance(quantity, int) or quantity <= 0:
            raise InvalidQuantityError()

        self.open()
        get_inventory_query = '''
            SELECT ProductID, QuantityInStock
            FROM Inventory
            WHERE InventoryID = %s
        '''
        self.stmt.execute(get_inventory_query, (inventory_id,))
        inventory_data = self.stmt.fetchone()

        if not inventory_data:
            print(f'No data found for InventoryID: {inventory_id}')
            return

        product_id, current_quantity = inventory_data

        new_quantity = quantity
        update_inventory_query = '''
            UPDATE Inventory
            SET QuantityInStock = %s
            WHERE InventoryID = %s
        '''

        self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
        self.conn.commit()
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```

O/P:



```
enter your choice9
enter inventory Id10
enter quantity15
--Database Is Connected--
Quantity Updated successfully
Connection Closed.
```



```
(9, 9, 10, datetime.datetime(2024, 4, 13, 11, 30, 54))
(10, 10, 15, datetime.datetime(2024, 4, 13, 11, 30, 54))
```

• **Product Search and Retrieval:**

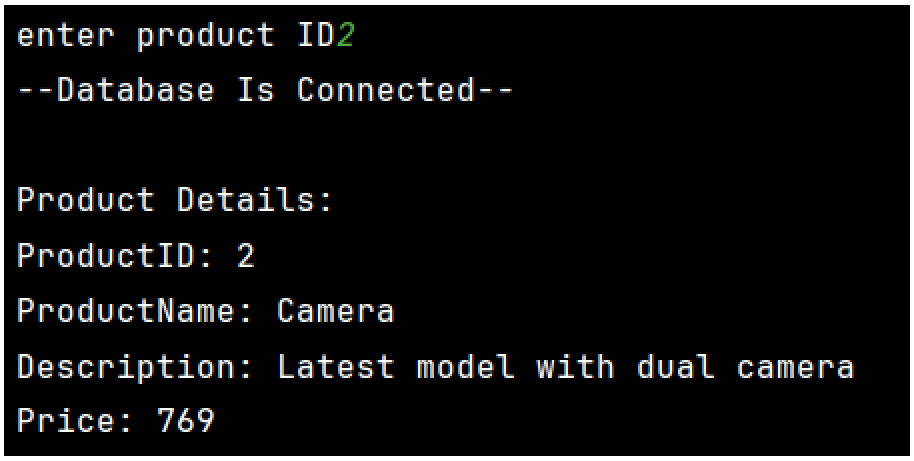
o Challenge: Implementing a search functionality to find products based on various criteria (e.g., name, category).

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

```
def GetProductDetails(self, product_id):
    try:
        # Validate product_id
        if not isinstance(product_id, int) or product_id < 0:
            raise InvalidIDError()

        self.open()
        get_product_details_str = '''
            SELECT * FROM Products
            WHERE Product_ID = %s
        '''
        self.stmt.execute(get_product_details_str, (product_id,))
        product_data = self.stmt.fetchone()

        if not product_data:
            print(f"No product found with ProductID: {product_id}")
        else:
            print('\nProduct Details:')
            print(f"ProductID: {product_data[0]}")
            print(f"ProductName: {product_data[1]}")
            print(f"Description: {product_data[2]}")
            print(f"Price: {product_data[3]}")
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```

O/P:



```
enter product ID2
--Database Is Connected--

Product Details:
ProductID: 2
ProductName: Camera
Description: Latest model with dual camera
Price: 769
```

- **Payment Records List:**

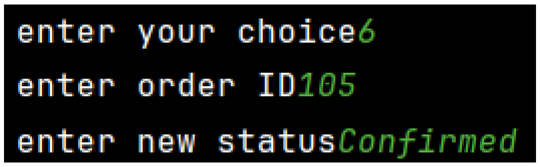
o Challenge: Managing a list of payment records for orders (List<PaymentClass>).

o Scenario: Recording and updating payment information for each order.

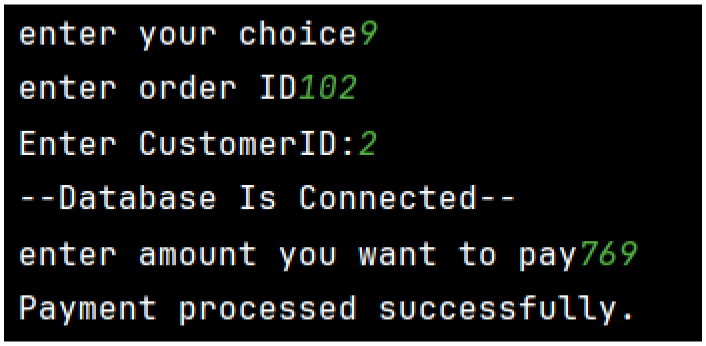
o Solution: Implement methods to record payments, update payment statuses, and handle payment errors. Ensure that payment records are consistent with order records.

```
class PaymentFailedException(Exception):
    def __init__(self, message="Payment Declined. Retry Again"):
        self.message = message
        super().__init__(self.message)
def UpdatePaymentStatus(self, order_id, new_status):
    if not isinstance(order_id, int) or order_id < 0:
        raise InvalidIDError()

    if not isinstance(new_status, str):
        raise CustomError("status should be string")
    self.selectOrders()
    update_str = 'UPDATE Orders SET Status = %s WHERE OrderID = %s'
    data = (new_status, order_id)
    self.open()
    self.stmt.execute(update_str, data)
    self.conn.commit()
    print('Order status updated successfully.')
    self.selectOrders()
    self.close()
def ProcessPayment(self,oid):
    CustomerID = int(input('Enter CustomerID:'))
    if not isinstance(CustomerID, int) or CustomerID < 0:
        raise InvalidIDError()
    self.CustomerId = CustomerID
    self.open()
    TotalAmount="SELECT TotalAmount from Orders where customerid=%s"

    self.stmt.execute(TotalAmount, (CustomerID,))
    records = self.stmt.fetchone()
    rec=list(records)
    totalamount=rec[0]
    try:

        amount = totalamount
        entered_amount = float(input("enter amount you want to pay"))
        if (entered_amount == amount):
            print("Payment processed successfully.")
        else:

            raise PaymentFailedException()
            self.ProcessPayment(oid)
    except PaymentFailedException as e:

        print(f"Error processing payment: {e}")
```

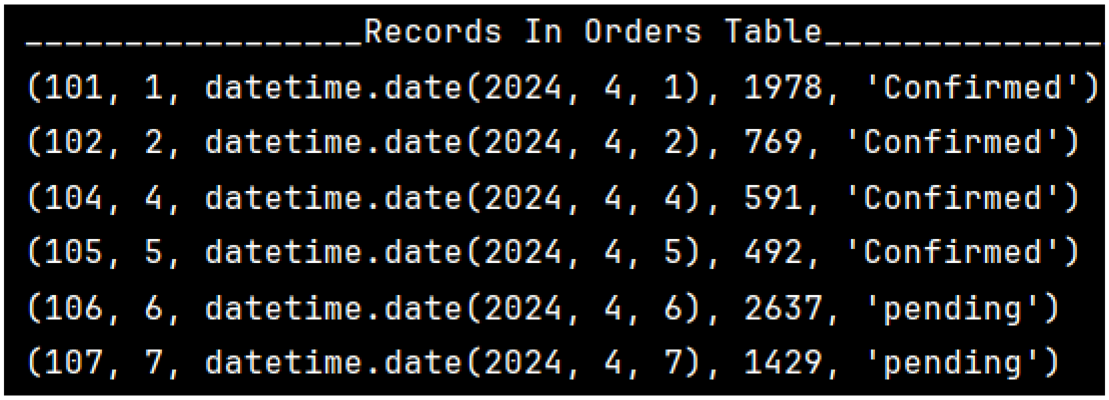
O/P:



```
enter your choice6
enter order ID105
enter new statusConfirmed
```



```
enter your choice9
enter order ID102
Enter CustomerID:2
--Database Is Connected--
enter amount you want to pay769
Payment processed successfully.
```



```
_________________Records In Orders Table______________
(101, 1, datetime.date(2024, 4, 1), 1978, 'Confirmed')
(102, 2, datetime.date(2024, 4, 2), 769, 'Confirmed')
(104, 4, datetime.date(2024, 4, 4), 591, 'Confirmed')
(105, 5, datetime.date(2024, 4, 5), 492, 'Confirmed')
(106, 6, datetime.date(2024, 4, 6), 2637, 'pending')
(107, 7, datetime.date(2024, 4, 7), 1429, 'pending')
```

- **OrderDetails and Products Relationship:**

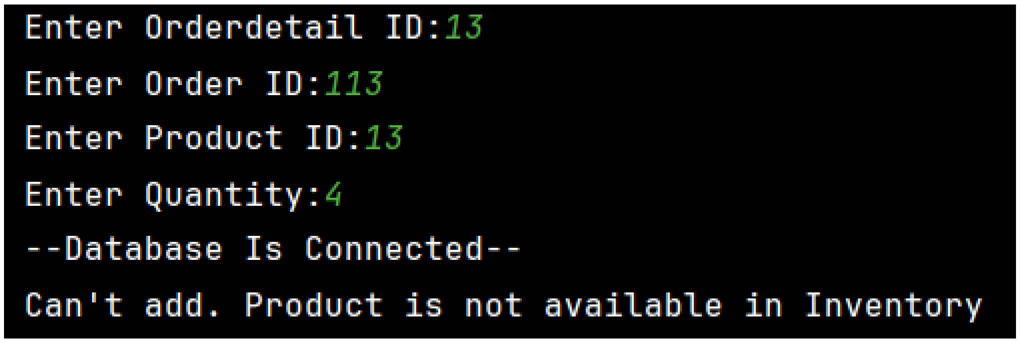
o Challenge: Managing the relationship between OrderDetails and Products.

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.

```
def addOrderDetail(self):
    OrderDetailID = int(input('Enter Orderdetail ID:'))
    if not isinstance(OrderDetailID, int) or OrderDetailID < 0:
        raise InvalidIDError()
    self.OrderDetailID = OrderDetailID

    OrderID = int(input('Enter Order ID:'))
    if not isinstance(OrderID, int) or OrderID < 0:
        raise InvalidIDError()
    self.OrderID = OrderID

    ProductID = int(input('Enter Product ID:'))
    if not isinstance(ProductID, int):
        raise InvalidIDError()
    self.ProductID = ProductID

    Quantity = int(input('Enter Quantity:'))
    if not isinstance(Quantity, int) or Quantity<0:
        raise InvalidQuantityError()
    self.Quantity = Quantity

    data = [(self.OrderDetailID,self.OrderID, self.ProductID, self.Quantity)]
    self.open()
    check_query='SELECT productid from inventory'
    self.stmt.execute(check_query)
    records = self.stmt.fetchall()
    a=list(records)
    numbers_list = [t[0] for t in a]

    if ProductID in numbers_list:

            insert_orderdetail_str = '''INSERT INTO OrderDetails (OrderDetailID, OrderID,
ProductID, Quantity)
                                        VALUES (%s, %s, %s, %s)'''

            self.open()
            self.stmt.executemany(insert_orderdetail_str, data)
            self.conn.commit()
            print('OrderDetail Record Inserted Successfully..')
            self.close()
    else:
        print("Can't add. Product is not available in Inventory")
```

O/P:



```
Enter Orderdetail ID:13
Enter Order ID:113
Enter Product ID:13
Enter Quantity:4
--Database Is Connected--
Can't add. Product is not available in Inventory
```

Task 7: Database Connectivity

• Implement a DatabaseConnector class responsible for establishing a connection to the "TechShopDB" database. This class should include methods for opening, closing, and managing database connections.

• Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.

1: Customer Registration

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

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

```python
def addCustomer(self):

    first_name = input('Enter First Name :')
    if not isinstance(first_name, str):
        raise InvalidNameError()
    StringCheck(first_name)
    self.firstname = first_name

    last_name = input('Enter Last Name :')
    if not isinstance(last_name, str):
        raise InvalidNameError()
    StringCheck(last_name)
    self.lastname = last_name

    Email = input('Enter email:')
    if not isinstance(Email, str):
        raise InvalidEmailError()
    validate_email(Email)
    self.email = Email

    Phone = input('Enter phone :')
    if not isinstance(Phone, str):
        raise InvalidPhoneError()
    validate_phone(Phone)
    self.phone = Phone

    Address = input('Enter address :')
    self.address = Address

    numberoforders = input('Enter orders :')
    self.numberoforders = numberoforders

    data = [(self.firstname, self.lastname, self.email, self.phone, self.address,
self.numberoforders)]
    insert_str = '''insert into
Customers(FirstName,LastName,Email,Phone,Address,numberoforders)
    values(%s,%s,%s,%s,%s,%s)'''
    self.open()
    self.stmt.executemany(insert_str, data)
    self.conn.commit()
    print('Records Inserted Successfully..')
    self.close()
```

2: Product Catalog Management

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

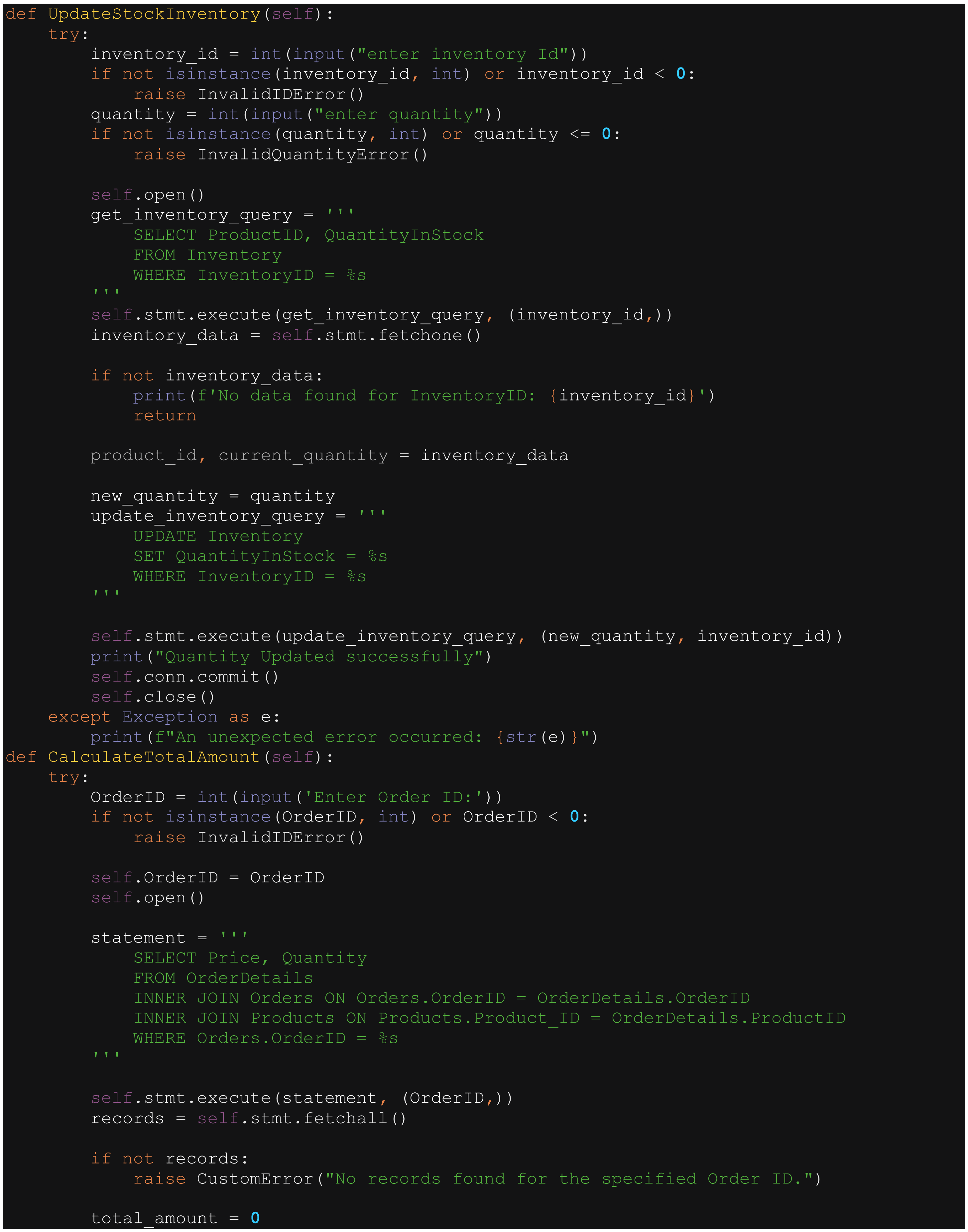
Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency

```python
def UpdateProductInfo(self):
    self.selectProducts()
    Product_id = int(input('Input Product ID to be Updated: '))
    if not isinstance(Product_id, int) or Product_id<0:
        raise InvalidIDError()
    Id = Product_id
    update_str = 'UPDATE Products SET '
    data = []

    Product_name = input('Enter Product Name ((Press Enter to skip)):')
    if Product_name:
        if not isinstance(Product_name, str):
            raise InvalidNameError()
        StringCheck(Product_name)
        self.ProductName = Product_name
        update_str += 'ProductName=%s, '
        data.append(self.ProductName)

    Description = input('Enter Description:(Press Enter to skip)')
    if Description:
        self.Description = Description
        update_str += 'Description=%s, '
        data.append(self.Description)

    Price = input('Enter Price: (Press Enter to skip): ')
    if Price:
        Price = int(Price)
        if not isinstance(Price, (int, float)) or Price < 0:
            raise InvalidPriceError()
        self.Price = int(Price)
        update_str += 'Price=%s, '
        data.append(self.Price)

    update_str = update_str.rstrip(', ')

    update_str += ' WHERE Product_ID=%s'
    data.append(Id)

    self.open()
    self.stmt.execute(update_str, data)
    self.conn.commit()
    print('Record updated successfully.')
    self.selectProducts()
```

3. Placing Customer Orders

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

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



```python
def UpdateStockInventory(self):
    try:
        inventory_id = int(input("enter inventory Id"))
        if not isinstance(inventory_id, int) or inventory_id < 0:
            raise InvalidIDError()
        quantity = int(input("enter quantity"))
        if not isinstance(quantity, int) or quantity <= 0:
            raise InvalidQuantityError()

        self.open()
        get_inventory_query = '''
            SELECT ProductID, QuantityInStock
            FROM Inventory
            WHERE InventoryID = %s
        '''
        self.stmt.execute(get_inventory_query, (inventory_id,))
        inventory_data = self.stmt.fetchone()

        if not inventory_data:
            print(f'No data found for InventoryID: {inventory_id}')
            return

        product_id, current_quantity = inventory_data

        new_quantity = quantity
        update_inventory_query = '''
            UPDATE Inventory
            SET QuantityInStock = %s
            WHERE InventoryID = %s
        '''

        self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
        print("Quantity Updated successfully")
        self.conn.commit()
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
def CalculateTotalAmount(self):
    try:
        OrderID = int(input('Enter Order ID:'))
        if not isinstance(OrderID, int) or OrderID < 0:
            raise InvalidIDError()

        self.OrderID = OrderID
        self.open()

        statement = '''
            SELECT Price, Quantity
            FROM OrderDetails
            INNER JOIN Orders ON Orders.OrderID = OrderDetails.OrderID
            INNER JOIN Products ON Products.Product_ID = OrderDetails.ProductID
            WHERE Orders.OrderID = %s
        '''

        self.stmt.execute(statement, (OrderID,))
        records = self.stmt.fetchall()

        if not records:
            raise CustomError("No records found for the specified Order ID.")

        total_amount = 0
        for record in records:
            price = float(record[0])
            quantity = int(record[1])
            total_amount += price * quantity

        discount = float(input("Enter discount (in percentage):"))

        if discount < 0 or discount > 100:
            raise CustomError("discount should be between 0-100")

        discount /= 100
        total_amount *= (1 - discount)
        print(total_amount)

        update_statement = 'UPDATE Orders SET TotalAmount=%s WHERE OrderID=%s'
        update_data = (Decimal(total_amount), OrderID)

        self.stmt.execute(update_statement, update_data)
        self.conn.commit()
        self.close()
        print("Total Amount after discount:", total_amount)

    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```

4: Tracking Order Status

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

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

```
def GetOrderDetails(self, order_id):
    try:
        if not isinstance(order_id, int) or order_id < 0:
            raise InvalidIDError()
        query = '''
            SELECT OrderID, CustomerID, OrderDate, TotalAmount, Status
            FROM Orders
            WHERE OrderID = %s
        '''

        self.open()
        self.stmt.execute(query, (order_id,))
        order_details = self.stmt.fetchone()

        if order_details:
            print('\nOrder Details:')
            print(f"OrderID: {order_details[0]}")
            print(f"CustomerID: {order_details[1]}")
            print(f"OrderDate: {order_details[2]}")
            print(f"TotalAmount: {order_details[3]}")
            print(f"Status: {order_details[4]}")
        else:
            print(f'Order with OrderID {order_id} not found.')

    except ValueError as ve:
        print(f"Error: {ve}")

    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")

    finally:
        self.close()
```

5: Inventory Management

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

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

```
def AddToInventory(self):
    try:
        inventory_id = int(input("enter inventory Id:"))
        if not isinstance(inventory_id, int) or inventory_id < 0:
            raise InvalidIDError()
        quantity = int(input("enter quantity to be added: "))
        if not isinstance(quantity, int) or quantity <= 0:
            raise InvalidQuantityError()

        self.open()
        get_inventory_query = '''
            SELECT ProductID, QuantityInStock
            FROM Inventory
            WHERE InventoryID = %s
        '''
        self.stmt.execute(get_inventory_query, (inventory_id,))
        inventory_data = self.stmt.fetchone()

        if not inventory_data:
            print(f'No data found for InventoryID: {inventory_id}')
            return

        product_id, current_quantity = inventory_data

        new_quantity = current_quantity + quantity

        update_inventory_query = '''
            UPDATE Inventory
            SET QuantityInStock = %s
            WHERE InventoryID = %s
        '''

        self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
        print(f"{quantity} units added Successfully")
        self.conn.commit()
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
def UpdateStockInventory(self):
    try:
        inventory_id = int(input("enter inventory Id"))
        if not isinstance(inventory_id, int) or inventory_id < 0:
            raise InvalidIDError()
        quantity = int(input("enter quantity"))
        if not isinstance(quantity, int) or quantity <= 0:
            raise InvalidQuantityError()

        self.open()
        get_inventory_query = '''
```

```
            SELECT ProductID, QuantityInStock
            FROM Inventory
            WHERE InventoryID = %s
        '''
        self.stmt.execute(get_inventory_query, (inventory_id,))
        inventory_data = self.stmt.fetchone()

        if not inventory_data:
            print(f'No data found for InventoryID: {inventory_id}')
            return

        product_id, current_quantity = inventory_data

        new_quantity = quantity
        update_inventory_query = '''
            UPDATE Inventory
            SET QuantityInStock = %s
            WHERE InventoryID = %s
        '''

        self.stmt.execute(update_inventory_query, (new_quantity, inventory_id))
        print("Quantity Updated successfully")
        self.conn.commit()
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```

6: Sales Reporting

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

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

```
def SalesReporting(self):
    self.open()
    select_orderdetails_str = '''SELECT ProductID,SUM(Quantity) FROM OrderDetails group
by ProductID'''
    self.stmt.execute(select_orderdetails_str)
    records = self.stmt.fetchall()
    print('')
    print('_________________Sales Report________________________')
    for record in records:
        print(f"Product ID is {record[0]} sold {int(record[1])} units")
    self.close()
```

7: Customer Account Updates

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

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

```
def UpdateCustomerInfo(self):
    self.select()
    customer_id = int(input('Input Customer ID to be Updated: '))
    if not isinstance(customer_id, int) :
        raise InvalidIDError()
    else:
        if customer_id < 0:
            raise InvalidIDError()
    Id = customer_id
    update_str = 'UPDATE customers SET '
    data = []

    first_name = input('Enter First Name ((Press Enter to skip)):')
    if first_name:
        if not isinstance(first_name, str):
            raise InvalidNameError()
        StringCheck(first_name)
        self.firstname = first_name
        update_str += 'FirstName=%s, '
        data.append(self.firstname)

    last_name = input('Enter Last Name ((Press Enter to skip)):')
    if last_name:
        if not isinstance(last_name, str):
            raise InvalidNameError()
        StringCheck(last_name)
        self.lastname = last_name
        update_str += 'LastName=%s, '
        data.append(self.lastname)

    Email = input('Enter email:(Press Enter to skip)')
    if Email:
        if not isinstance(Email, str):
            raise InvalidEmailError()
        validate_email(Email)
        self.email = Email
        update_str += 'Email=%s, '
        data.append(self.email)

    Phone = input('Enter Phone (Press Enter to skip): ')
    if Phone:
        if not isinstance(Phone, str):
            raise InvalidPhoneError()
        validate_phone(Phone)
        self.phone = Phone
        update_str += 'Phone=%s, '
        data.append(self.phone)

    update_str = update_str.rstrip(', ')

    update_str += ' WHERE CustomerID=%s'
    data.append(Id)

    self.open()
    self.stmt.execute(update_str, data)
    self.conn.commit()
    print('Record updated successfully.')
    self.select()
```

8: Payment Processing

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

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

```
def ProcessPayment(self,oid):
    CustomerID = int(input('Enter CustomerID:'))
    if not isinstance(CustomerID, int) or CustomerID < 0:
        raise InvalidIDError()
    self.CustomerId = CustomerID


    self.open()
    TotalAmount="SELECT TotalAmount from Orders where customerid=%s"

    self.stmt.execute(TotalAmount, (CustomerID,))
    records = self.stmt.fetchone()
    rec=list(records)
    totalamount=rec[0]
    try:

        amount = totalamount
        entered_amount = float(input("enter amount you want to pay"))
        if (entered_amount == amount):
            print("Payment processed successfully.")
            update_str = 'UPDATE Orders SET Status = %s WHERE CustomerID = %s'
            data = ("Confirmed",self.CustomerId)
            self.open()
            self.stmt.execute(update_str, data)
            self.conn.commit()
            print('Order status updated successfully.')
            self.selectOrders()
            self.close()
        else:

            raise PaymentFailedException()
            self.ProcessPayment(oid)
    except PaymentFailedException as e:

        print(f"Error processing payment: {e}")
```

9: Product Search and Recommendations

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

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

```
def GetProductDetails(self, product_id):
    try:
        # Validate product_id
        if not isinstance(product_id, int) or product_id < 0:
            raise InvalidIDError()

        self.open()
        get_product_details_str = '''
            SELECT * FROM Products
            WHERE Product_ID = %s
        '''
        self.stmt.execute(get_product_details_str, (product_id,))
        product_data = self.stmt.fetchone()

        if not product_data:
            print(f"No product found with ProductID: {product_id}")
```

```python
        else:
            print('\nProduct Details:')
            print(f"ProductID: {product_data[0]}")
            print(f"ProductName: {product_data[1]}")
            print(f"Description: {product_data[2]}")
            print(f"Price: {product_data[3]}")
        self.close()
    except Exception as e:
        print(f"An unexpected error occurred: {str(e)}")
```