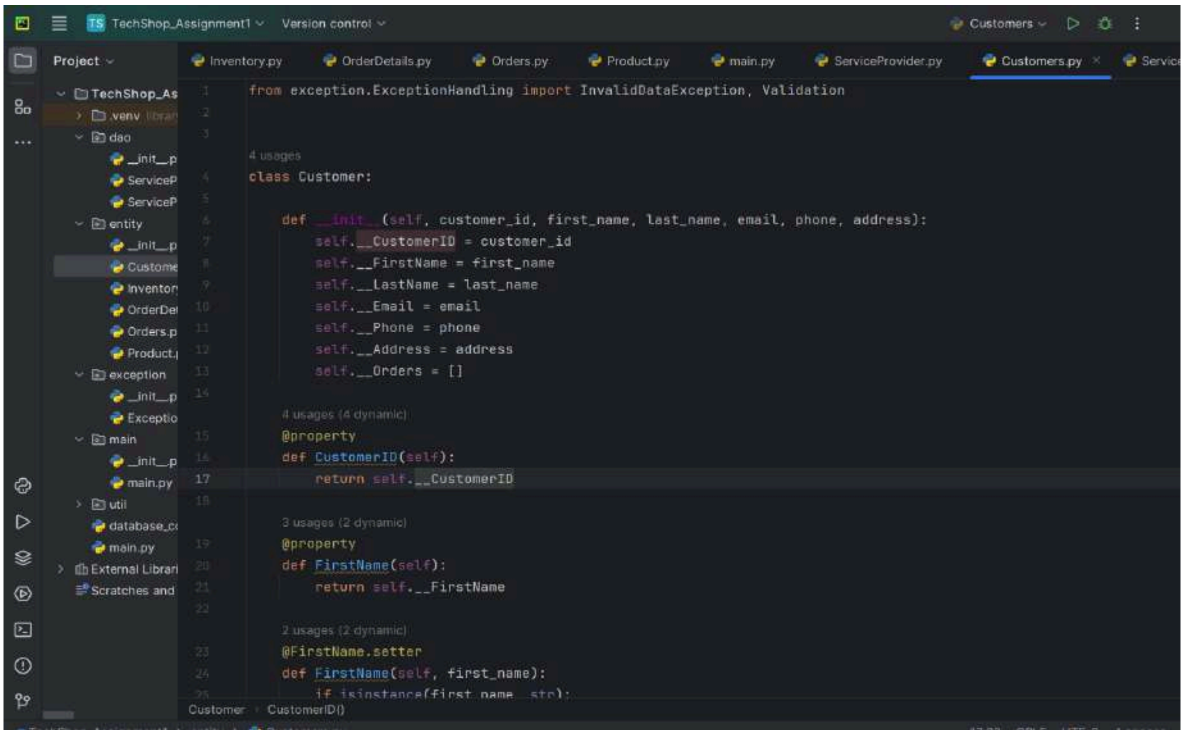
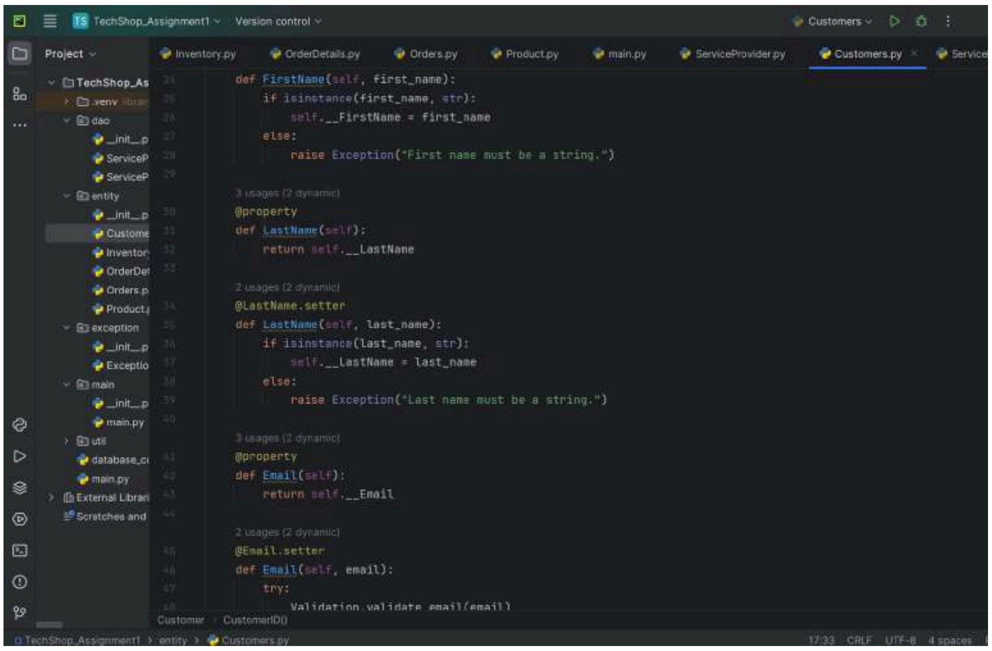
Assignment 1: TechShop, an electronic gadgets shop

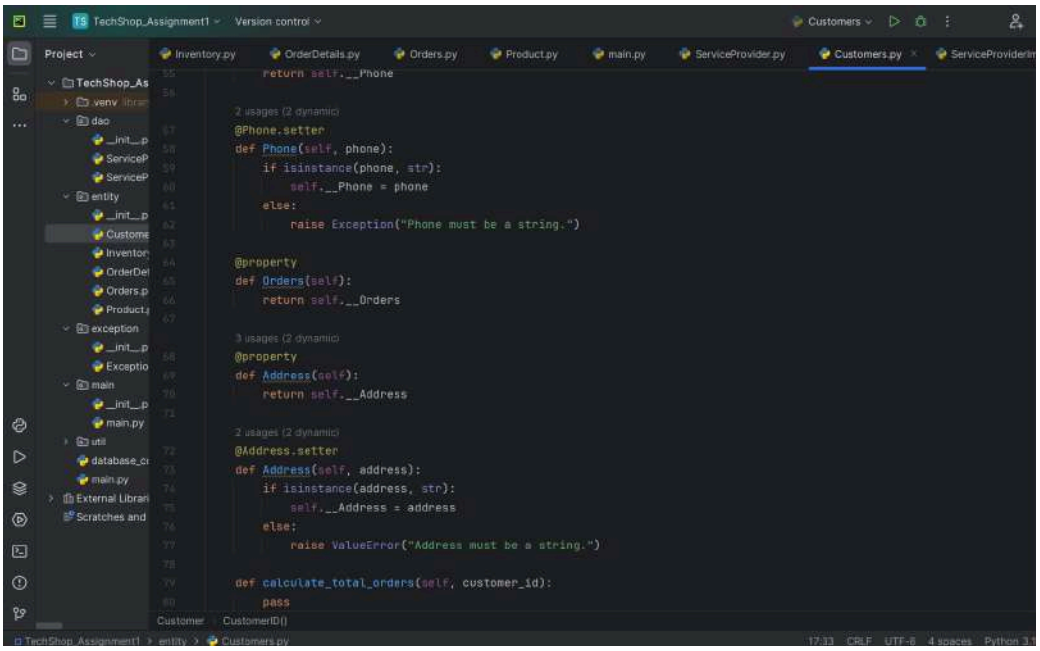
Implement OOPs Task 1:
Classes and Their Attributes:
You are working as a software developer for TechShop, a company that sells electronic gadgets. Your task is to design and implement an application using Object-Oriented Programming (OOP) principles to manage customer information, product details, and orders. Below are the classes you need to create:
Customers Class:
 Attributes:
• CustomerID (int) • FirstName (string) • LastName (string) • Email (string) • Phone (string) • Address (string)
Methods:
• CalculateTotalOrders(): Calculates the total number of orders placed by this customer. • GetCustomerDetails(): Retrieves and displays detailed information about the customer. • UpdateCustomerInfo(): Allows the customer to update their information (e.g., email, phone, or address).



```python
from exception.ExceptionHandling import InvalidDataException, Validation


class Customer:

    def __init__(self, customer_id, first_name, last_name, email, phone, address):
        self.__CustomerID = customer_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__Phone = phone
        self.__Address = address
        self.__Orders = []

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

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

    @FirstName.setter
    def FirstName(self, first_name):
        if isinstance(first_name, str):
```

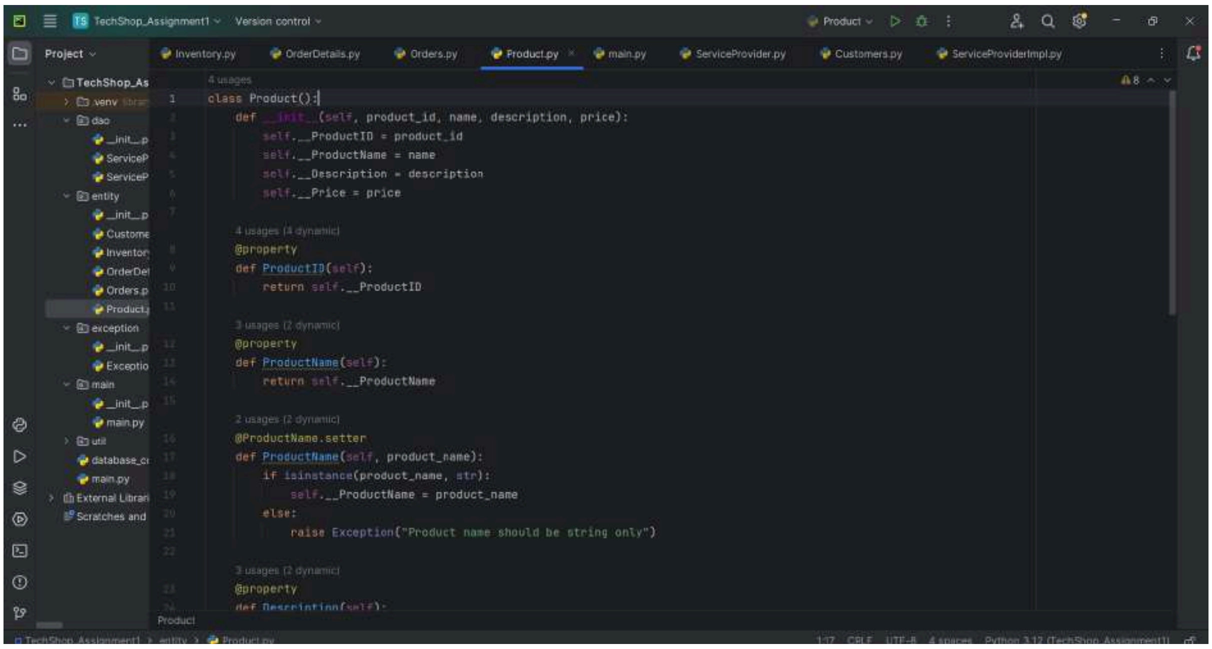


```python
    def FirstName(self, first_name):
        if isinstance(first_name, str):
            self.__FirstName = first_name
        else:
            raise Exception("First name must be a string.")

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

    @LastName.setter
    def LastName(self, last_name):
        if isinstance(last_name, str):
            self.__LastName = last_name
        else:
            raise Exception("Last name must be a string.")

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

    @Email.setter
    def Email(self, email):
        try:
            Validation.validate_email(email)
```

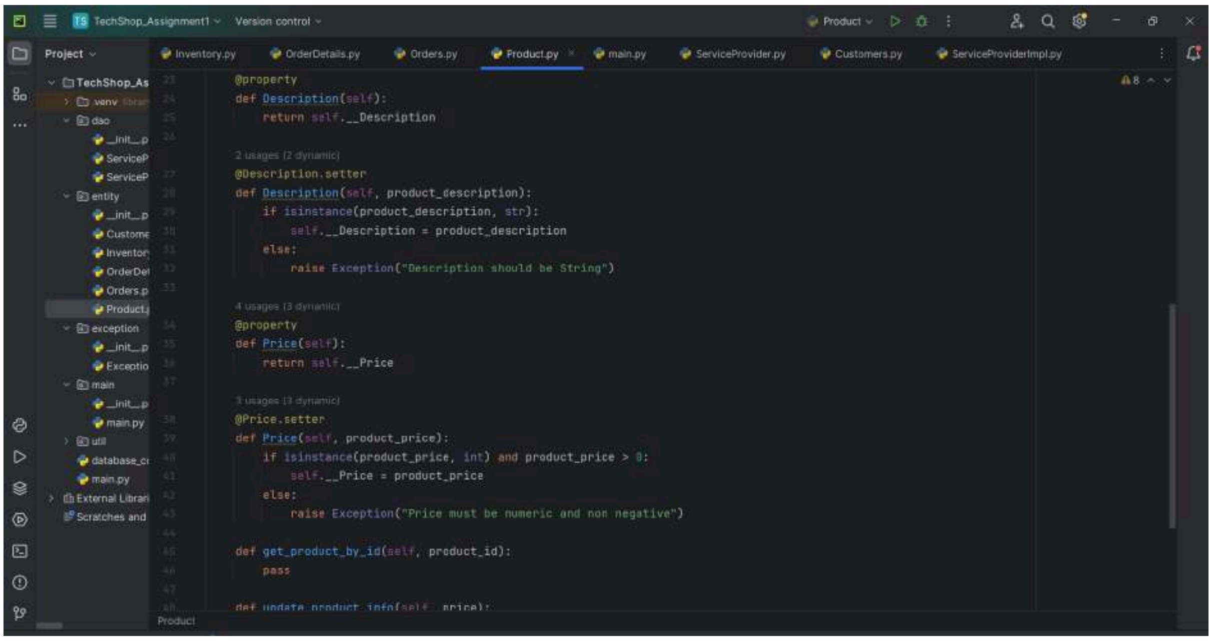


```python
        return self.__Phone

    @Phone.setter
    def Phone(self, phone):
        if isinstance(phone, str):
            self.__Phone = phone
        else:
            raise Exception("Phone must be a string.")

    @property
    def Orders(self):
        return self.__Orders

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

    @Address.setter
    def Address(self, address):
        if isinstance(address, str):
            self.__Address = address
        else:
            raise ValueError("Address must be a string.")

    def calculate_total_orders(self, customer_id):
        pass
```

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.



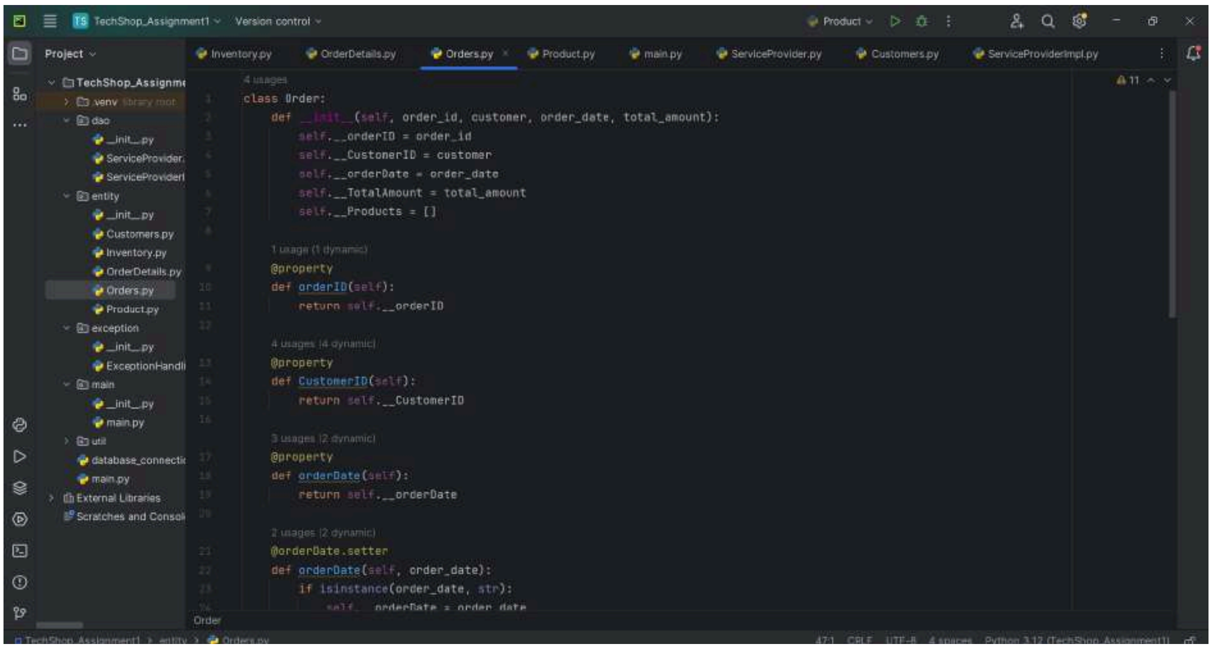
```python
class Product():
    def __init__(self, product_id, name, description, price):
        self.__ProductID = product_id
        self.__ProductName = name
        self.__Description = description
        self.__Price = price

    @property
    def ProductID(self):
        return self.__ProductID

    @property
    def ProductName(self):
        return self.__ProductName

    @ProductName.setter
    def ProductName(self, product_name):
        if isinstance(product_name, str):
            self.__ProductName = product_name
        else:
            raise Exception("Product name should be string only")

    @property
    def Description(self):
```



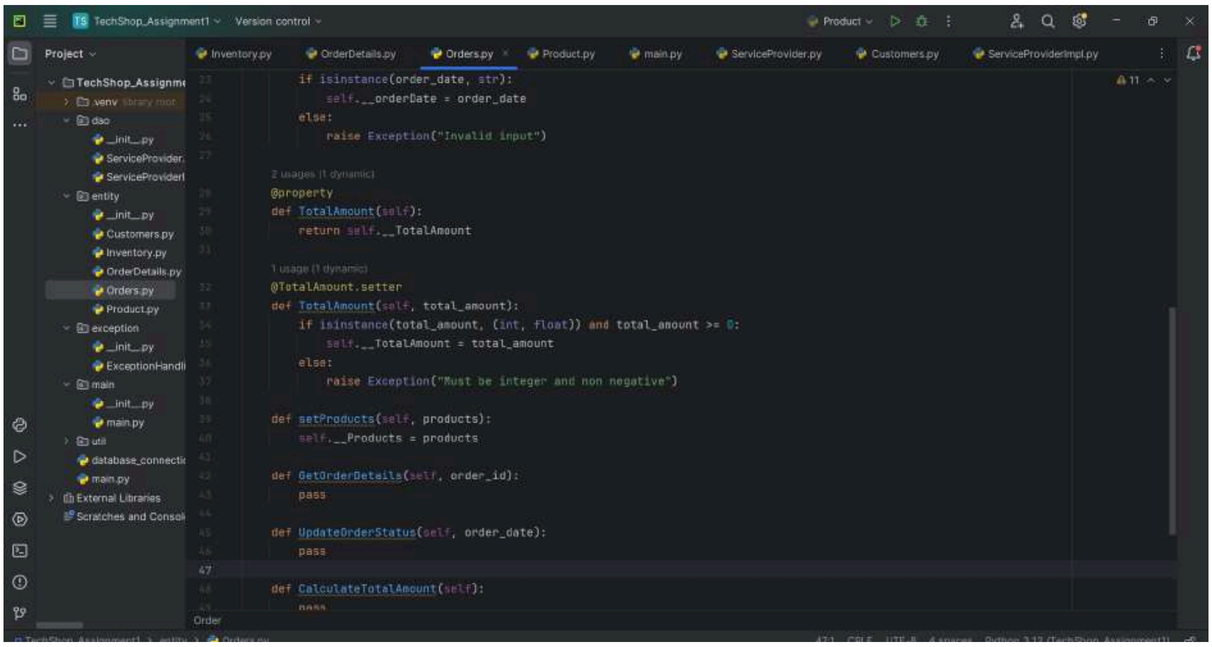
```python
    @property
    def Description(self):
        return self.__Description

    @Description.setter
    def Description(self, product_description):
        if isinstance(product_description, str):
            self.__Description = product_description
        else:
            raise Exception("Description should be String")

    @property
    def Price(self):
        return self.__Price

    @Price.setter
    def Price(self, product_price):
        if isinstance(product_price, int) and product_price > 0:
            self.__Price = product_price
        else:
            raise Exception("Price must be numeric and non negative")

    def get_product_by_id(self, product_id):
        pass

    def update_product_info(self, price):
```

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.



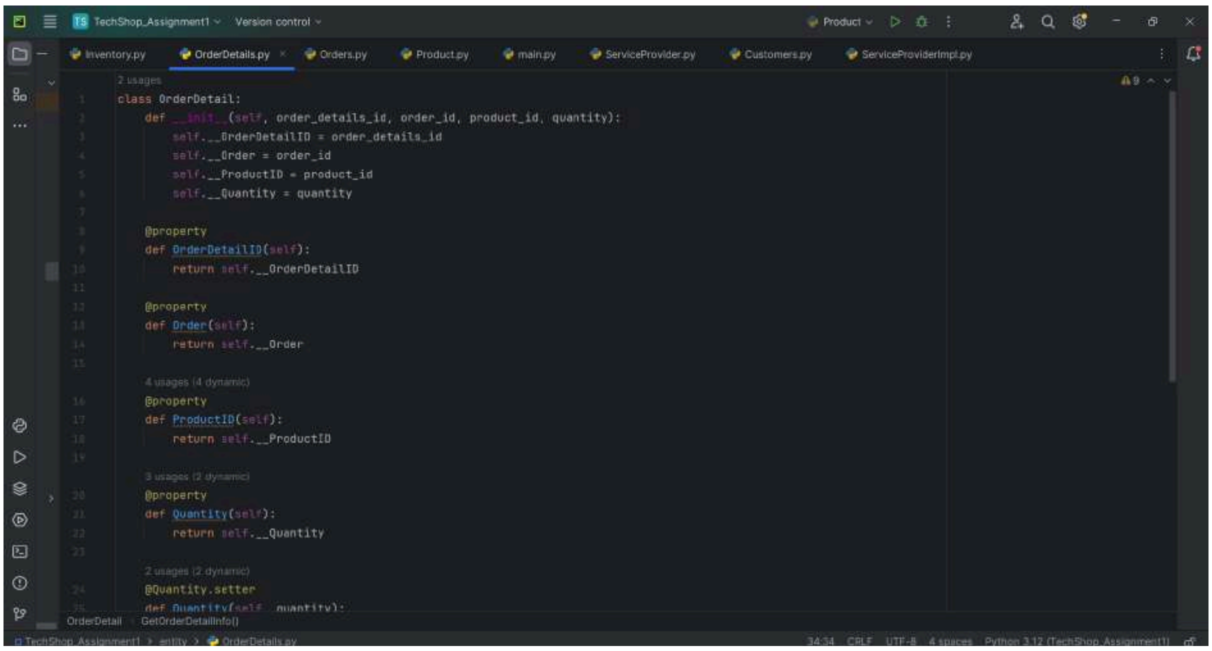
```python
class Order:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.__orderID = order_id
        self.__CustomerID = customer
        self.__orderDate = order_date
        self.__TotalAmount = total_amount
        self.__Products = []

    @property
    def orderID(self):
        return self.__orderID

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

    @property
    def orderDate(self):
        return self.__orderDate

    @orderDate.setter
    def orderDate(self, order_date):
        if isinstance(order_date, str):
            self.__orderDate = order_date
```



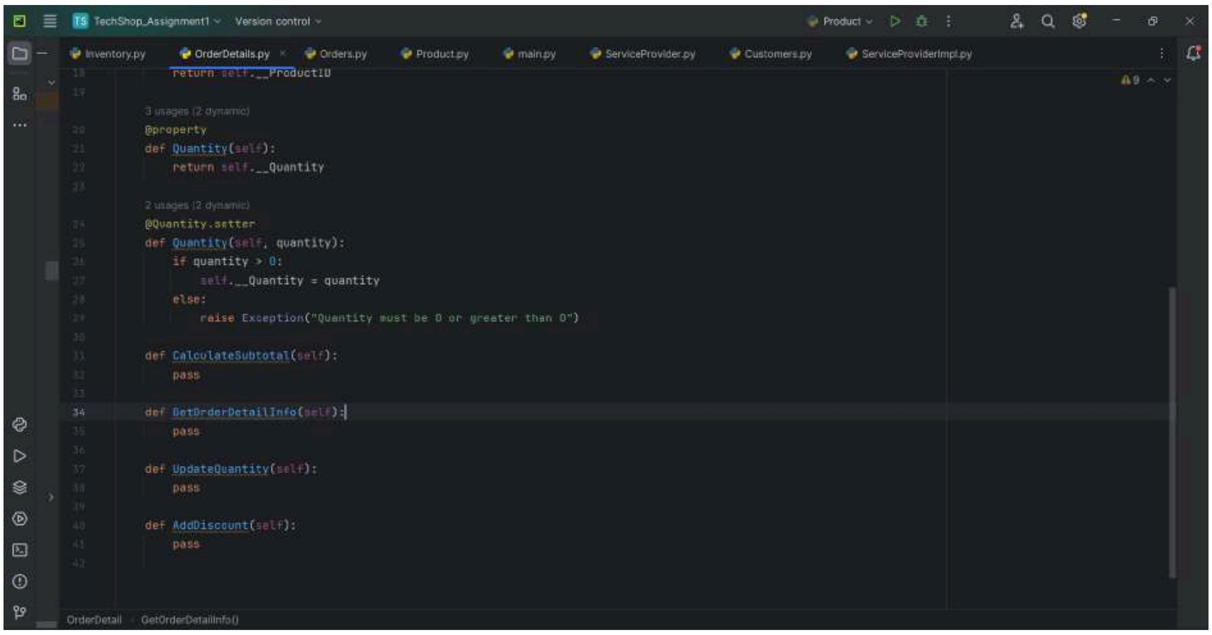
```python
        if isinstance(order_date, str):
            self.__orderDate = order_date
        else:
            raise Exception("Invalid input")

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

    @TotalAmount.setter
    def TotalAmount(self, total_amount):
        if isinstance(total_amount, (int, float)) and total_amount >= 0:
            self.__TotalAmount = total_amount
        else:
            raise Exception("Must be integer and non negative")

    def setProducts(self, products):
        self.__Products = products

    def GetOrderDetails(self, order_id):
        pass

    def UpdateOrderStatus(self, order_date):
        pass

    def CalculateTotalAmount(self):
        pass
```

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.



```python
class OrderDetail:
    def __init__(self, order_details_id, order_id, product_id, quantity):
        self.__OrderDetailID = order_details_id
        self.__Order = order_id
        self.__ProductID = product_id
        self.__Quantity = quantity

    @property
    def OrderDetailID(self):
        return self.__OrderDetailID

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

    @property
    def ProductID(self):
        return self.__ProductID

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

    @Quantity.setter
    def Quantity(self, quantity):
```

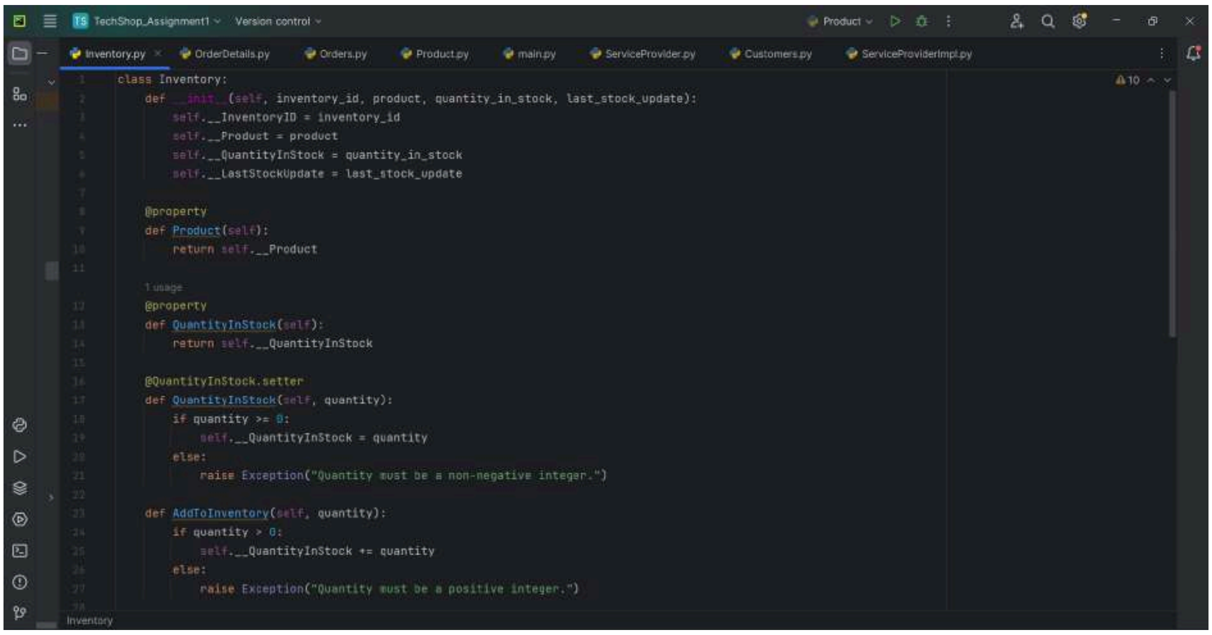


```python
        return self.__ProductID

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

    @Quantity.setter
    def Quantity(self, quantity):
        if quantity > 0:
            self.__Quantity = quantity
        else:
            raise Exception("Quantity must be 0 or greater than 0")

    def CalculateSubtotal(self):
        pass

    def GetOrderDetailInfo(self):
        pass

    def UpdateQuantity(self):
        pass

    def AddDiscount(self):
        pass
```

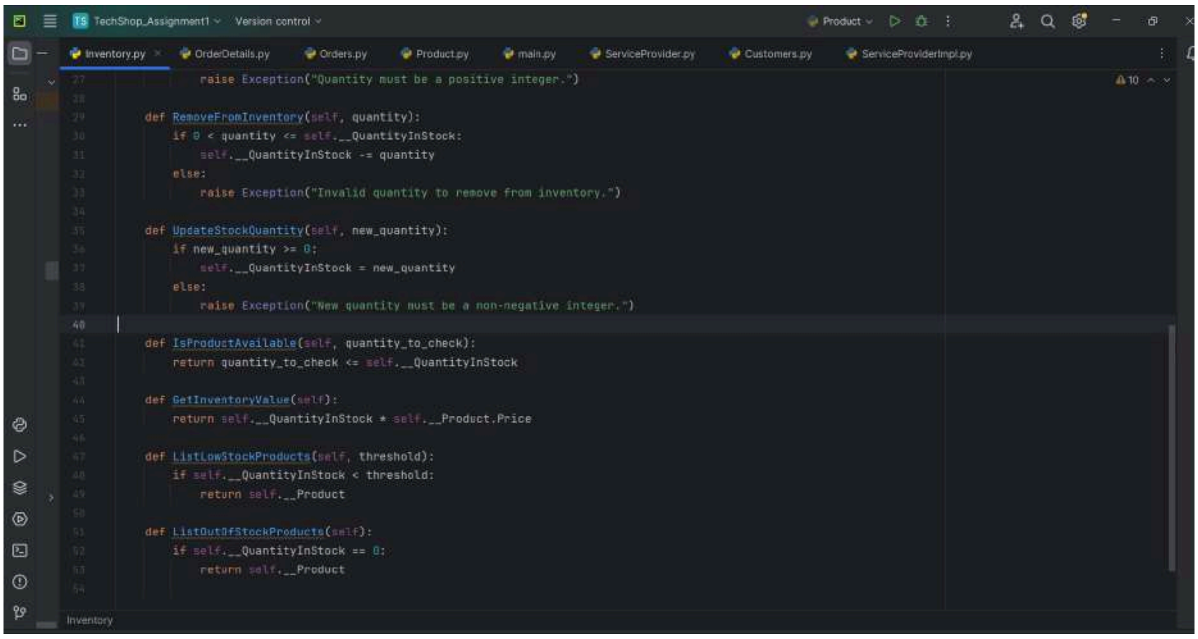
Inventory class:
Attributes:
• InventoryID(int) • Product (Composition): The product associated with the inventory item. • QuantityInStock: The quantity of the product currently in stock. • LastStockUpdate

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.



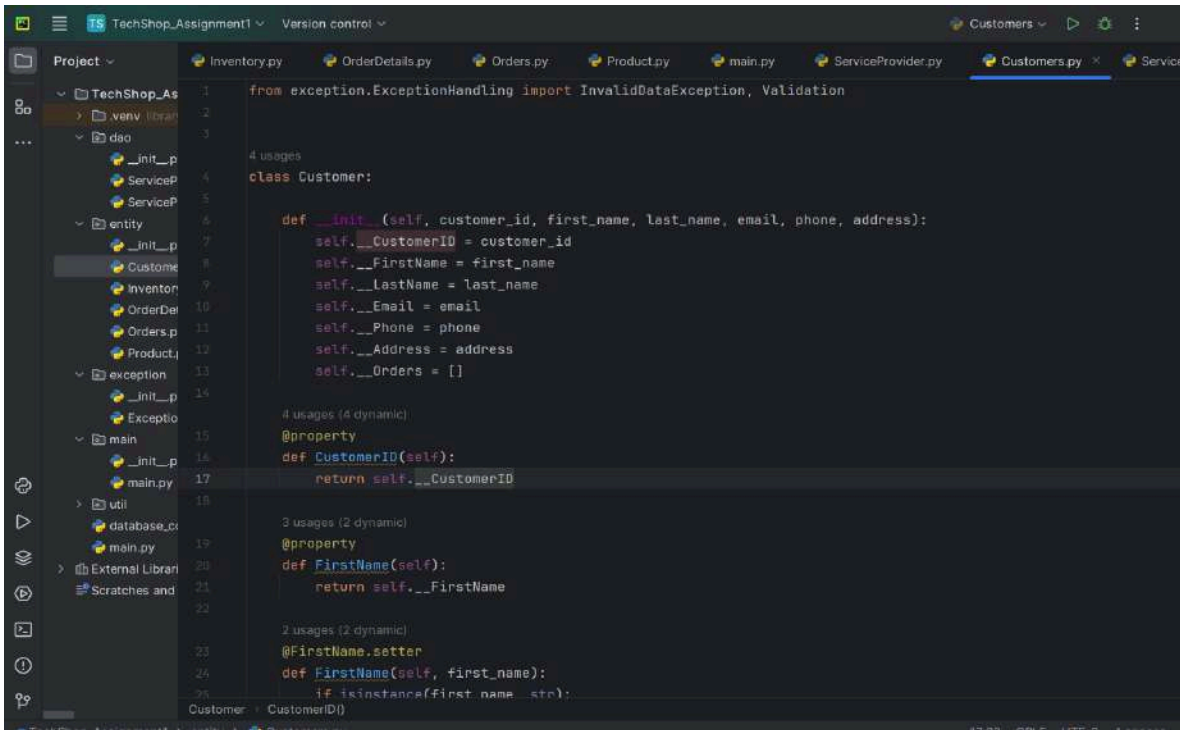
```python
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
        self.__InventoryID = inventory_id
        self.__Product = product
        self.__QuantityInStock = quantity_in_stock
        self.__LastStockUpdate = last_stock_update

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

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

    @QuantityInStock.setter
    def QuantityInStock(self, quantity):
        if quantity >= 0:
            self.__QuantityInStock = quantity
        else:
            raise Exception("Quantity must be a non-negative integer.")

    def AddToInventory(self, quantity):
        if quantity > 0:
            self.__QuantityInStock += quantity
        else:
            raise Exception("Quantity must be a positive integer.")
```



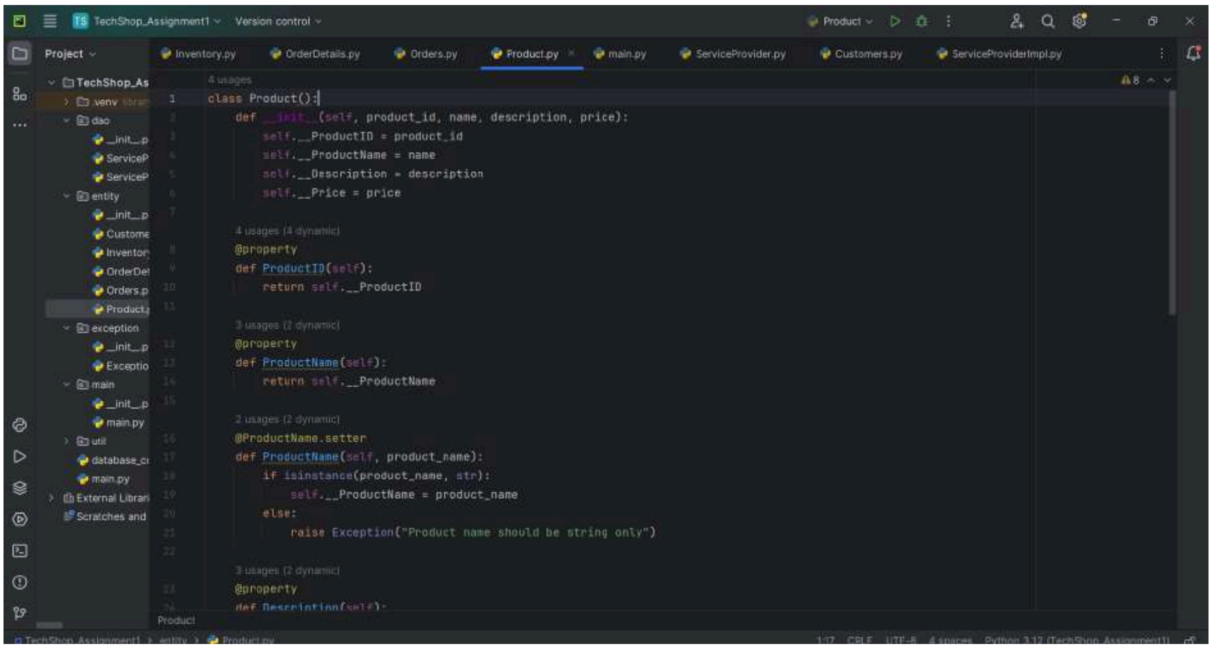
```python
                raise Exception("Quantity must be a positive integer.")

    def RemoveFromInventory(self, quantity):
        if 0 < quantity <= self.__QuantityInStock:
            self.__QuantityInStock -= quantity
        else:
            raise Exception("Invalid quantity to remove from inventory.")

    def UpdateStockQuantity(self, new_quantity):
        if new_quantity >= 0:
            self.__QuantityInStock = new_quantity
        else:
            raise Exception("New quantity must be a non-negative integer.")

    def IsProductAvailable(self, quantity_to_check):
        return quantity_to_check <= self.__QuantityInStock

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

    def ListLowStockProducts(self, threshold):
        if self.__QuantityInStock < threshold:
            return self.__Product

    def ListOutOfStockProducts(self):
        if self.__QuantityInStock == 0:
            return self.__Product
```

Task 2: Class Creation:

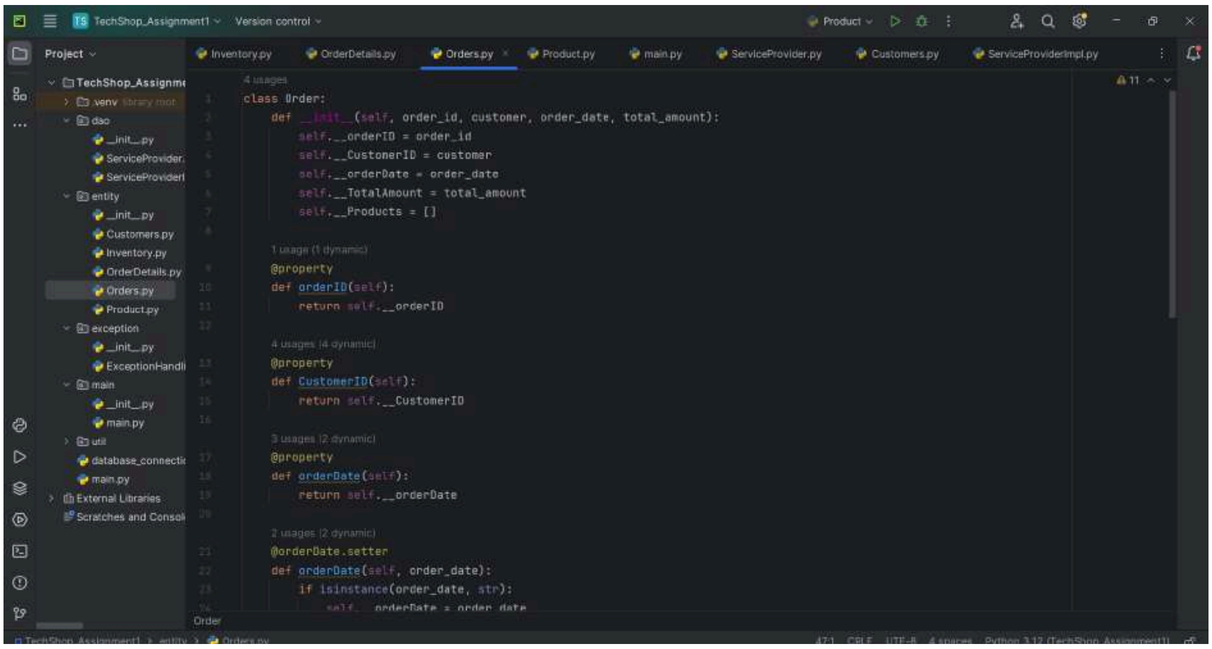
• Create the classes (Customers, Products, Orders, OrderDetails and Inventory) with the specified attributes. • Implement the constructor for each class to initialize its attributes. • Implement methods as specified.



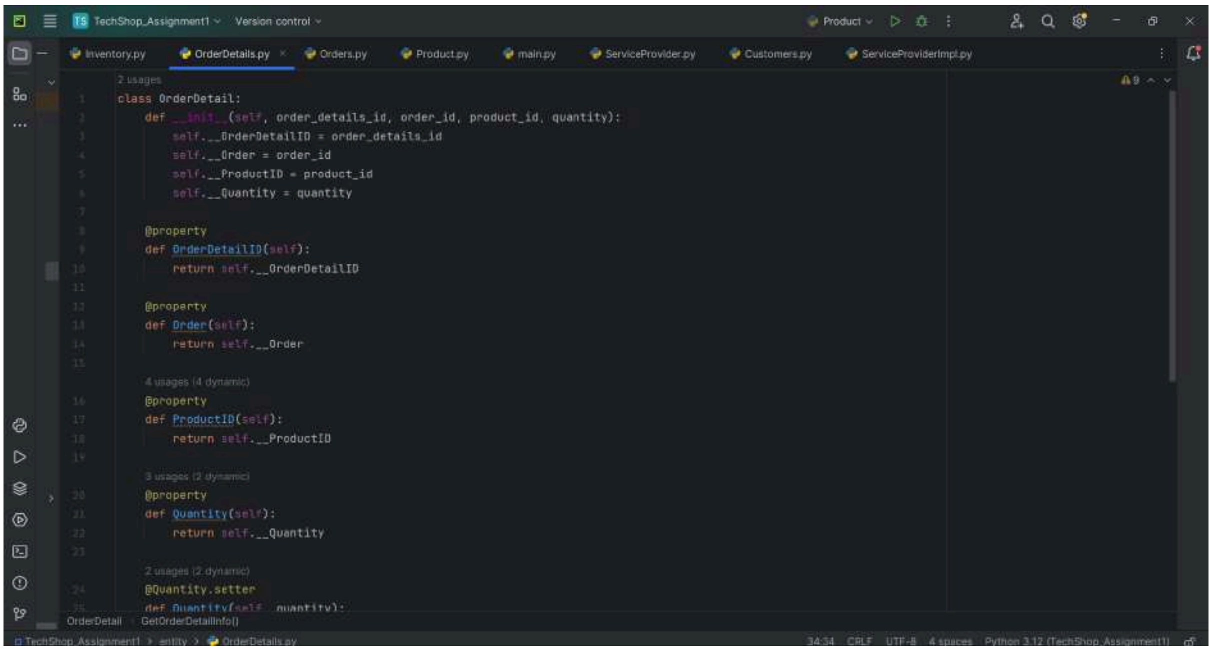
```python
from exception.ExceptionHandling import InvalidDataException, Validation


4 usages
class Customer:

    def __init__(self, customer_id, first_name, last_name, email, phone, address):
        self.__CustomerID = customer_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__Phone = phone
        self.__Address = address
        self.__Orders = []

    4 usages (4 dynamic)
    @property
    def CustomerID(self):
        return self.__CustomerID

    3 usages (2 dynamic)
    @property
    def FirstName(self):
        return self.__FirstName

    2 usages (2 dynamic)
    @FirstName.setter
    def FirstName(self, first_name):
        if isinstance(first_name, str):
```



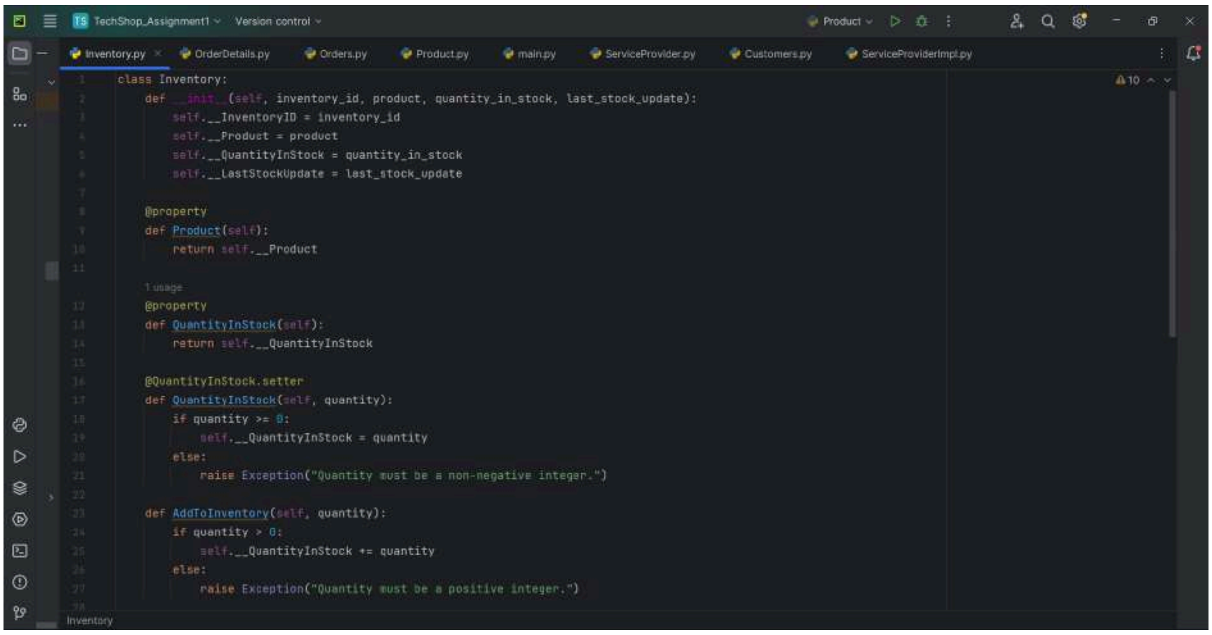
```python
class Product():
    def __init__(self, product_id, name, description, price):
        self.__ProductID = product_id
        self.__ProductName = name
        self.__Description = description
        self.__Price = price

    @property
    def ProductID(self):
        return self.__ProductID

    @property
    def ProductName(self):
        return self.__ProductName

    @ProductName.setter
    def ProductName(self, product_name):
        if isinstance(product_name, str):
            self.__ProductName = product_name
        else:
            raise Exception("Product name should be string only")

    @property
    def Description(self):
```



```python
class Order:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.__orderID = order_id
        self.__CustomerID = customer
        self.__orderDate = order_date
        self.__TotalAmount = total_amount
        self.__Products = []

    @property
    def orderID(self):
        return self.__orderID

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

    @property
    def orderDate(self):
        return self.__orderDate

    @orderDate.setter
    def orderDate(self, order_date):
        if isinstance(order_date, str):
            self.__orderDate = order_date
```



```python
class OrderDetail:
    def __init__(self, order_details_id, order_id, product_id, quantity):
        self.__OrderDetailID = order_details_id
        self.__Order = order_id
        self.__ProductID = product_id
        self.__Quantity = quantity

    @property
    def OrderDetailID(self):
        return self.__OrderDetailID

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

    @property
    def ProductID(self):
        return self.__ProductID

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

    @Quantity.setter
    def Quantity(self, quantity):
```



```python
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
        self.__InventoryID = inventory_id
        self.__Product = product
        self.__QuantityInStock = quantity_in_stock
        self.__LastStockUpdate = last_stock_update

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

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

    @QuantityInStock.setter
    def QuantityInStock(self, quantity):
        if quantity >= 0:
            self.__QuantityInStock = quantity
        else:
            raise Exception("Quantity must be a non-negative integer.")

    def AddToInventory(self, quantity):
        if quantity > 0:
            self.__QuantityInStock += quantity
        else:
            raise Exception("Quantity must be a positive integer.")
```

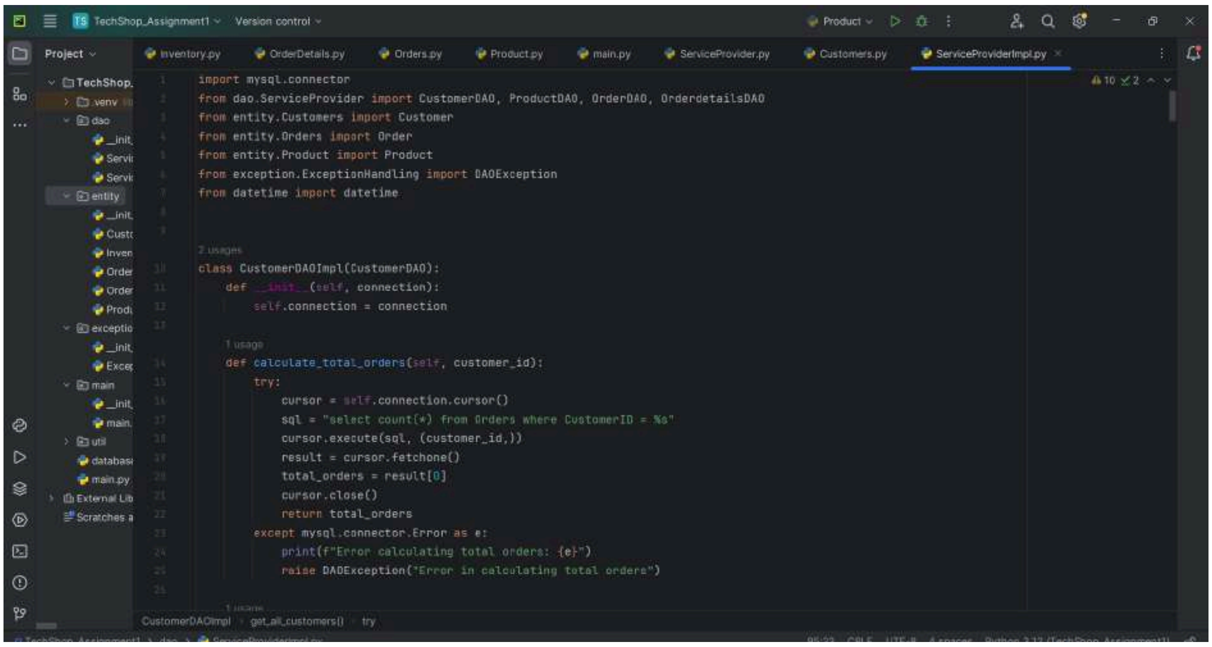
Task 3: Encapsulation:
• Implement encapsulation by making the attributes private and providing public properties (getters and setters) for each attribute.
• Add data validation logic to setter methods (e.g., ensure that prices are non-negative, quantities are positive integers).
Task 4:
Composition:
Ensure that the Order and OrderDetail classes correctly use composition to reference Customer and Product objects.
• Orders Class with Composition: o In the Orders class, we want to establish a composition relationship with the Customers class, indicating

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

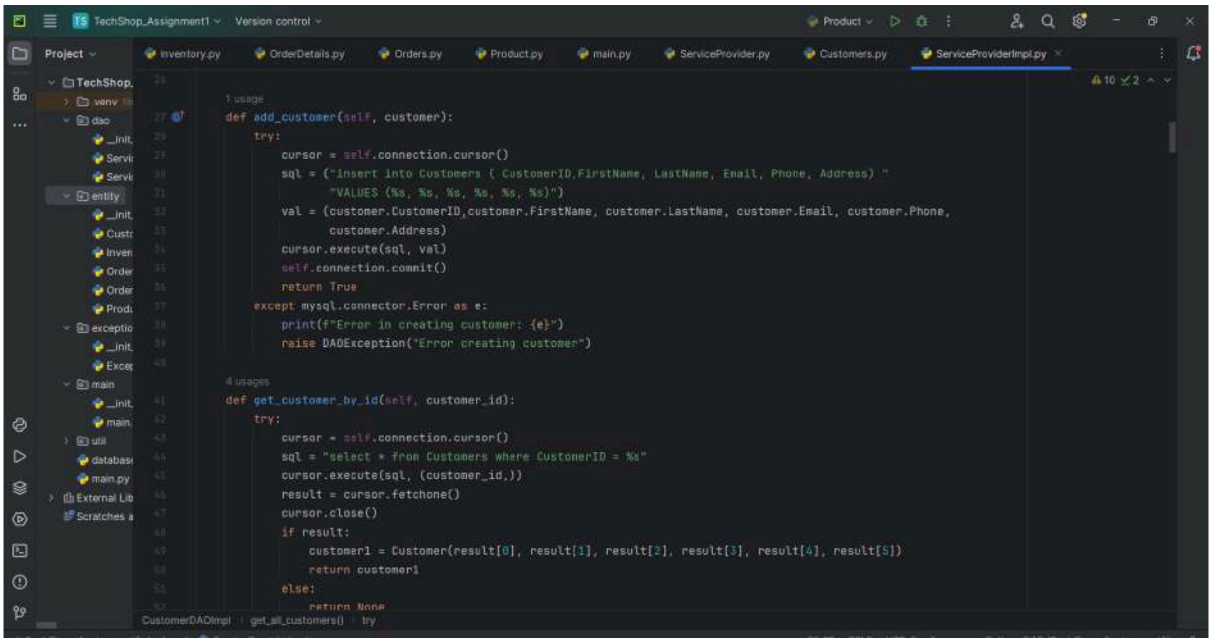
• OrderDetails Class with Composition: o Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered. o In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the product in the order detail.

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

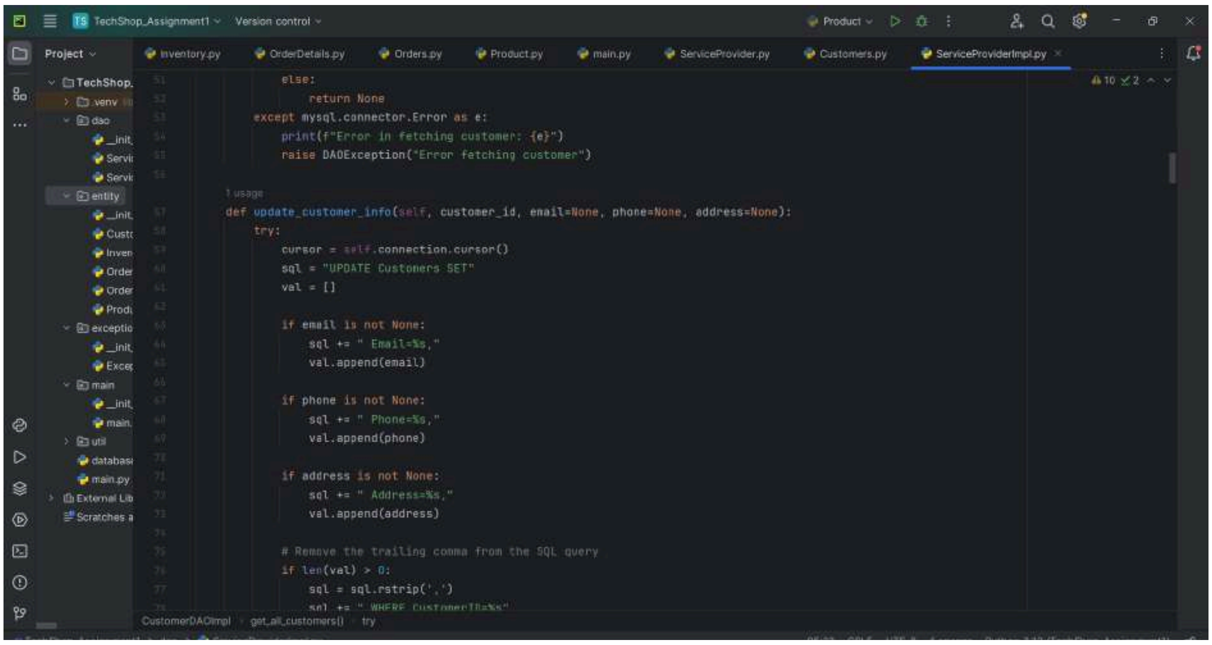
• Inventory Class: o The Inventory class represents the inventory of products available for sale. It can have composition relationships with the Products class to indicate which products are in the inventory.



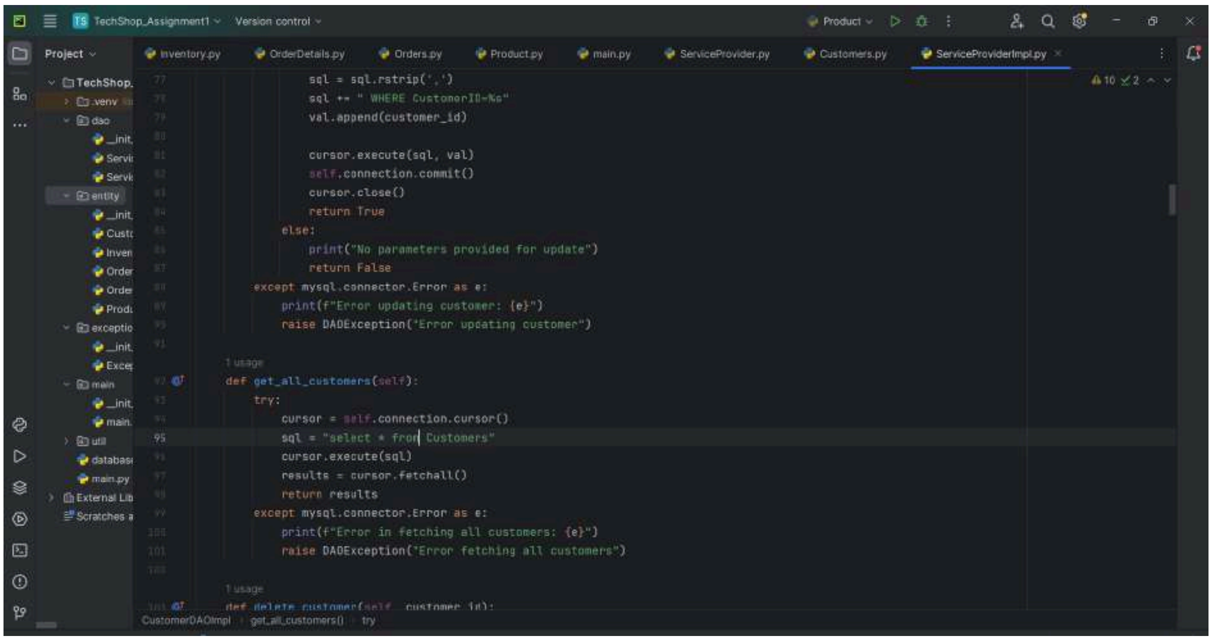
```
import mysql.connector
from dao.ServiceProvider import CustomerDAO, ProductDAO, OrderDAO, OrderdetailsDAO
from entity.Customers import Customer
from entity.Orders import Order
from entity.Product import Product
from exception.ExceptionHandling import DAOException
from datetime import datetime


class CustomerDAOImpl(CustomerDAO):
    def __init__(self, connection):
        self.connection = connection

    def calculate_total_orders(self, customer_id):
        try:
            cursor = self.connection.cursor()
            sql = "select count(*) from Orders where CustomerID = %s"
            cursor.execute(sql, (customer_id,))
            result = cursor.fetchone()
            total_orders = result[0]
            cursor.close()
            return total_orders
        except mysql.connector.Error as e:
            print(f"Error calculating total orders: {e}")
            raise DAOException("Error in calculating total orders")
```



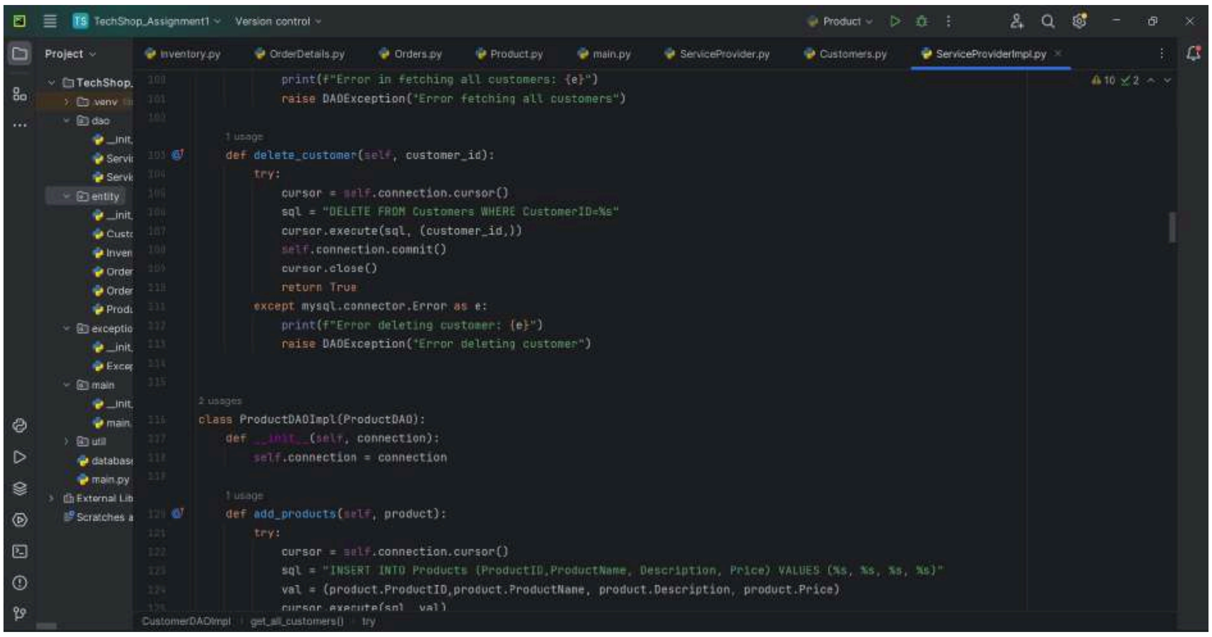
```
    1 usage
    def add_customer(self, customer):
        try:
            cursor = self.connection.cursor()
            sql = ("insert into Customers ( CustomerID,FirstName, LastName, Email, Phone, Address) "
                   "VALUES (%s, %s, %s, %s, %s, %s)")
            val = (customer.CustomerID,customer.FirstName, customer.LastName, customer.Email, customer.Phone,
                   customer.Address)
            cursor.execute(sql, val)
            self.connection.commit()
            return True
        except mysql.connector.Error as e:
            print(f"Error in creating customer: {e}")
            raise DAOException("Error creating customer")

    4 usages
    def get_customer_by_id(self, customer_id):
        try:
            cursor = self.connection.cursor()
            sql = "select * from Customers where CustomerID = %s"
            cursor.execute(sql, (customer_id,))
            result = cursor.fetchone()
            cursor.close()
            if result:
                customer1 = Customer(result[0], result[1], result[2], result[3], result[4], result[5])
                return customer1
            else:
                return None
```



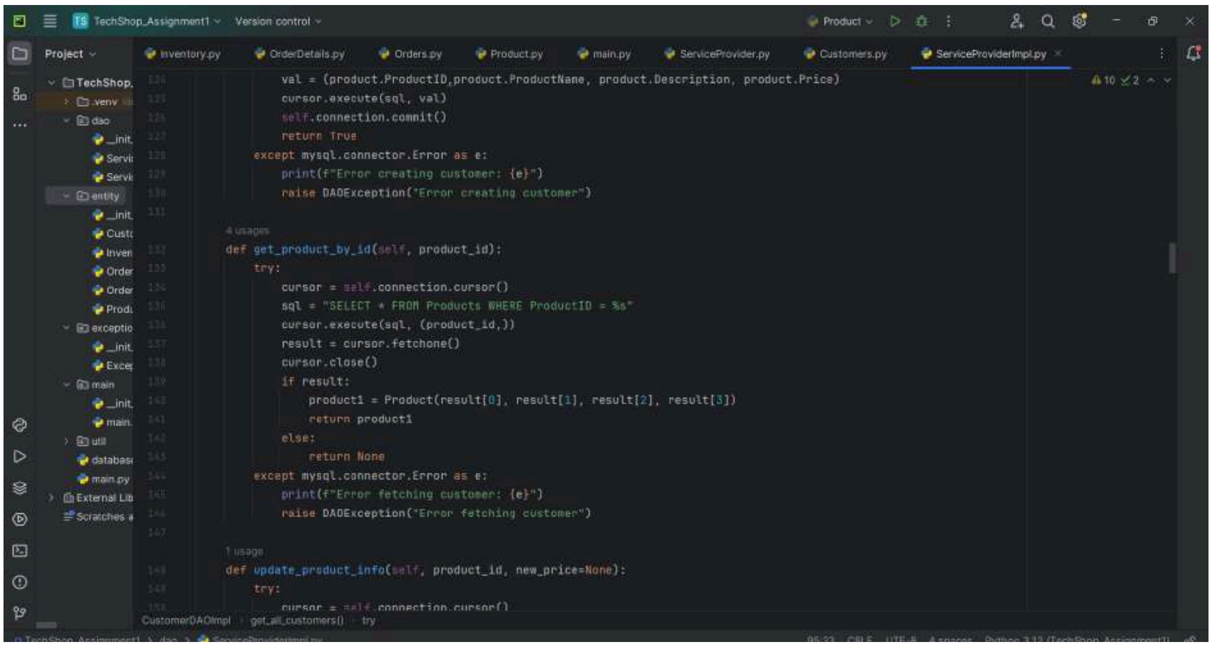
```
            else:
                return None
        except mysql.connector.Error as e:
            print(f"Error in fetching customer: {e}")
            raise DAOException("Error fetching customer")

    1 usage
    def update_customer_info(self, customer_id, email=None, phone=None, address=None):
        try:
            cursor = self.connection.cursor()
            sql = "UPDATE Customers SET"
            val = []

            if email is not None:
                sql += " Email=%s,"
                val.append(email)

            if phone is not None:
                sql += " Phone=%s,"
                val.append(phone)

            if address is not None:
                sql += " Address=%s,"
                val.append(address)

            # Remove the trailing comma from the SQL query
            if len(val) > 0:
                sql = sql.rstrip(',')
                sql += " WHERE CustomerID=%s"
```



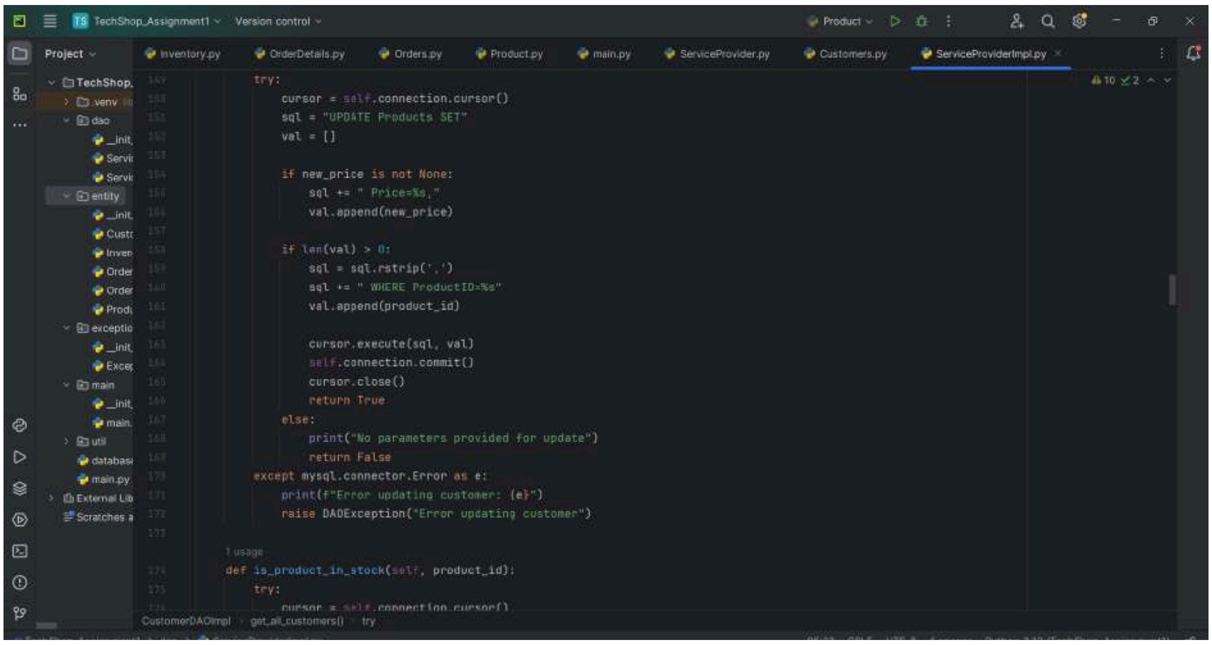
```
                sql = sql.rstrip(',')
                sql += " WHERE CustomerID=%s"
                val.append(customer_id)

                cursor.execute(sql, val)
                self.connection.commit()
                cursor.close()
                return True
            else:
                print("No parameters provided for update")
                return False
        except mysql.connector.Error as e:
            print(f"Error updating customer: {e}")
            raise DAOException("Error updating customer")

    1 usage
    def get_all_customers(self):
        try:
            cursor = self.connection.cursor()
            sql = "select * from Customers"
            cursor.execute(sql)
            results = cursor.fetchall()
            return results
        except mysql.connector.Error as e:
            print(f"Error in fetching all customers: {e}")
            raise DAOException("Error fetching all customers")

    1 usage
    def delete_customer(self, customer_id):
```



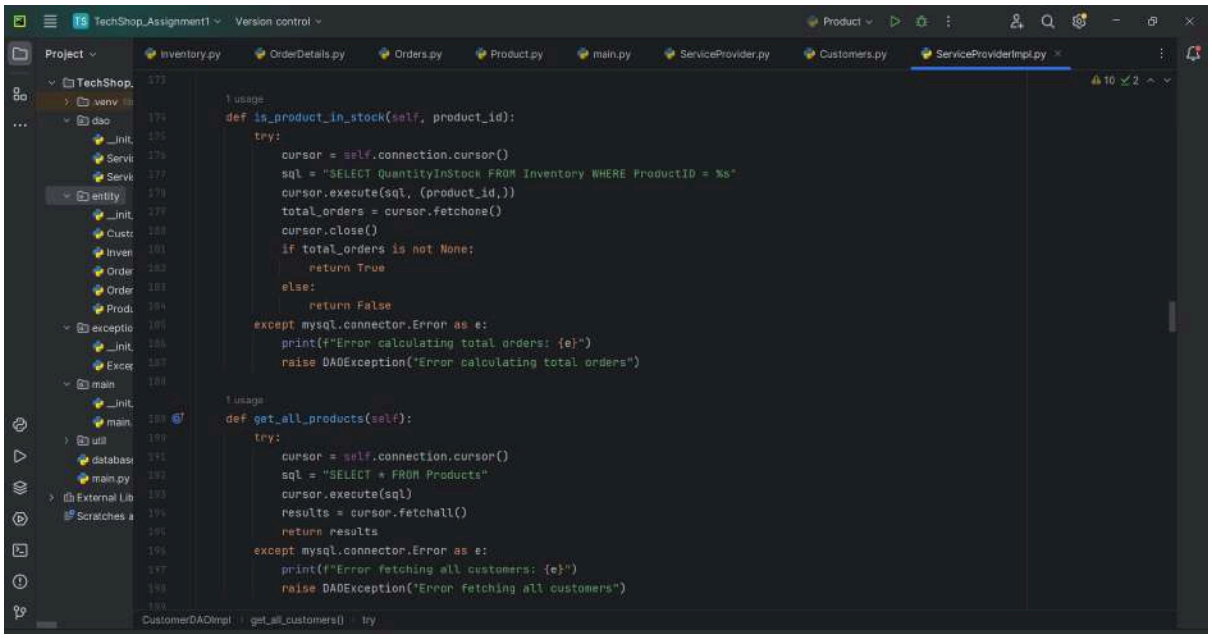
```
            print(f"Error in fetching all customers: {e}")
            raise DAOException("Error fetching all customers")

    1 usage
    def delete_customer(self, customer_id):
        try:
            cursor = self.connection.cursor()
            sql = "DELETE FROM Customers WHERE CustomerID=%s"
            cursor.execute(sql, (customer_id,))
            self.connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print(f"Error deleting customer: {e}")
            raise DAOException("Error deleting customer")


2 usages
class ProductDAOImpl(ProductDAO):
    def __init__(self, connection):
        self.connection = connection

    1 usage
    def add_products(self, product):
        try:
            cursor = self.connection.cursor()
            sql = "INSERT INTO Products (ProductID,ProductName, Description, Price) VALUES (%s, %s, %s, %s)"
            val = (product.ProductID,product.ProductName, product.Description, product.Price)
            cursor.execute(sql, val)
```



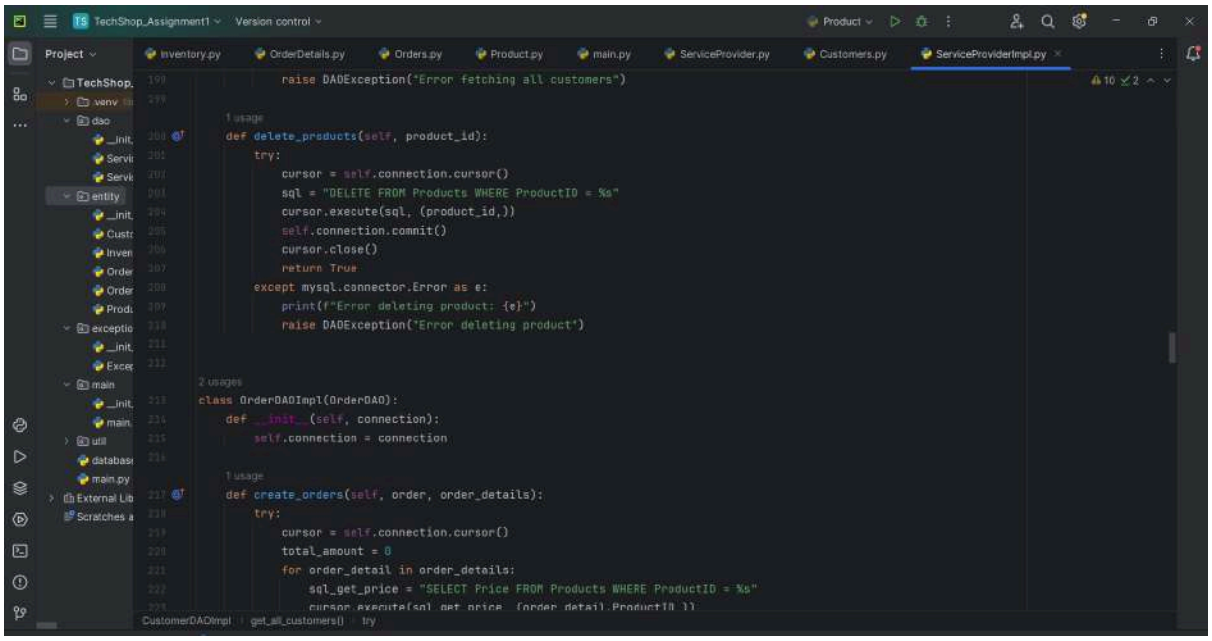
```python
            val = (product.ProductID,product.ProductName, product.Description, product.Price)
            cursor.execute(sql, val)
            self.connection.commit()
            return True
        except mysql.connector.Error as e:
            print(f"Error creating customer: {e}")
            raise DAOException("Error creating customer")

    4 usages
    def get_product_by_id(self, product_id):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT * FROM Products WHERE ProductID = %s"
            cursor.execute(sql, (product_id,))
            result = cursor.fetchone()
            cursor.close()
            if result:
                product1 = Product(result[0], result[1], result[2], result[3])
                return product1
            else:
                return None
        except mysql.connector.Error as e:
            print(f"Error fetching customer: {e}")
            raise DAOException("Error fetching customer")

    1 usage
    def update_product_info(self, product_id, new_price=None):
        try:
            cursor = self.connection.cursor()
```



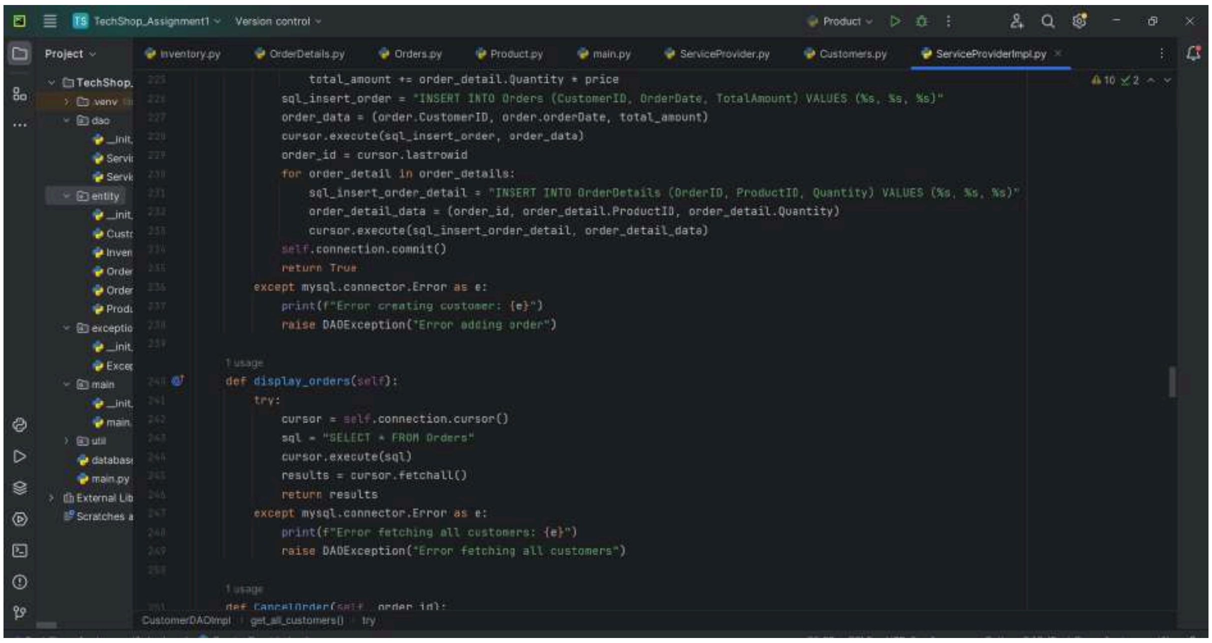
```python
        try:
            cursor = self.connection.cursor()
            sql = "UPDATE Products SET"
            val = []

            if new_price is not None:
                sql += " Price=%s,"
                val.append(new_price)

            if len(val) > 0:
                sql = sql.rstrip(',')
                sql += " WHERE ProductID=%s"
                val.append(product_id)

                cursor.execute(sql, val)
                self.connection.commit()
                cursor.close()
                return True
            else:
                print("No parameters provided for update")
                return False
        except mysql.connector.Error as e:
            print(f"Error updating customer: {e}")
            raise DAOException("Error updating customer")

    1 usage
    def is_product_in_stock(self, product_id):
        try:
            cursor = self.connection.cursor()
```



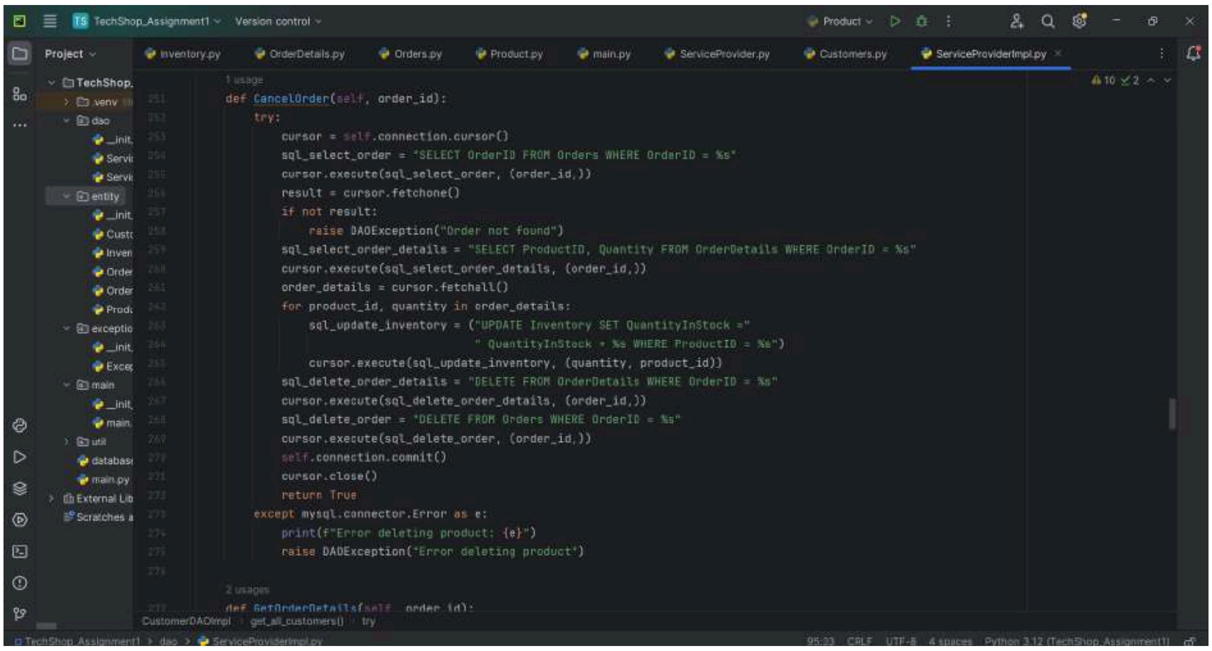
```
    def is_product_in_stock(self, product_id):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT QuantityInStock FROM Inventory WHERE ProductID = %s"
            cursor.execute(sql, (product_id,))
            total_orders = cursor.fetchone()
            cursor.close()
            if total_orders is not None:
                return True
            else:
                return False
        except mysql.connector.Error as e:
            print(f"Error calculating total orders: {e}")
            raise DAOException("Error calculating total orders")

    def get_all_products(self):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT * FROM Products"
            cursor.execute(sql)
            results = cursor.fetchall()
            return results
        except mysql.connector.Error as e:
            print(f"Error fetching all customers: {e}")
            raise DAOException("Error fetching all customers")
```



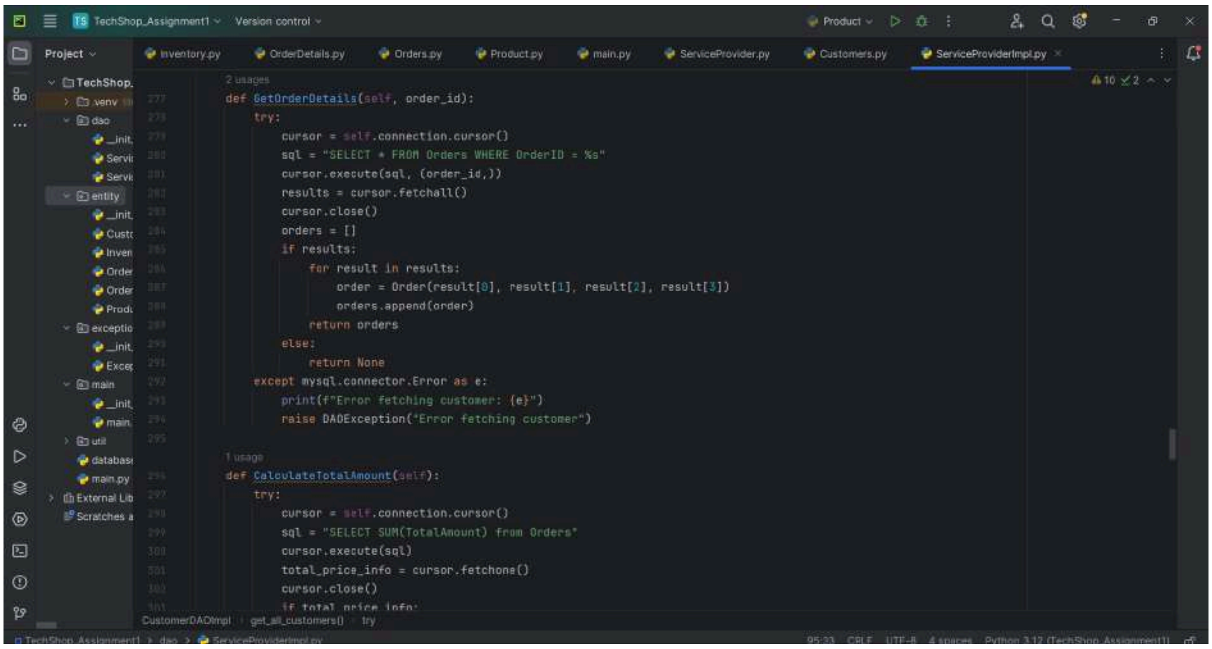
```
            raise DAOException("Error fetching all customers")

    def delete_products(self, product_id):
        try:
            cursor = self.connection.cursor()
            sql = "DELETE FROM Products WHERE ProductID = %s"
            cursor.execute(sql, (product_id,))
            self.connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print(f"Error deleting product: {e}")
            raise DAOException("Error deleting product")


class OrderDAOImpl(OrderDAO):
    def __init__(self, connection):
        self.connection = connection

    def create_orders(self, order, order_details):
        try:
            cursor = self.connection.cursor()
            total_amount = 0
            for order_detail in order_details:
                sql_get_price = "SELECT Price FROM Products WHERE ProductID = %s"
                cursor.execute(sql_get_price, (order_detail.ProductID,))
```



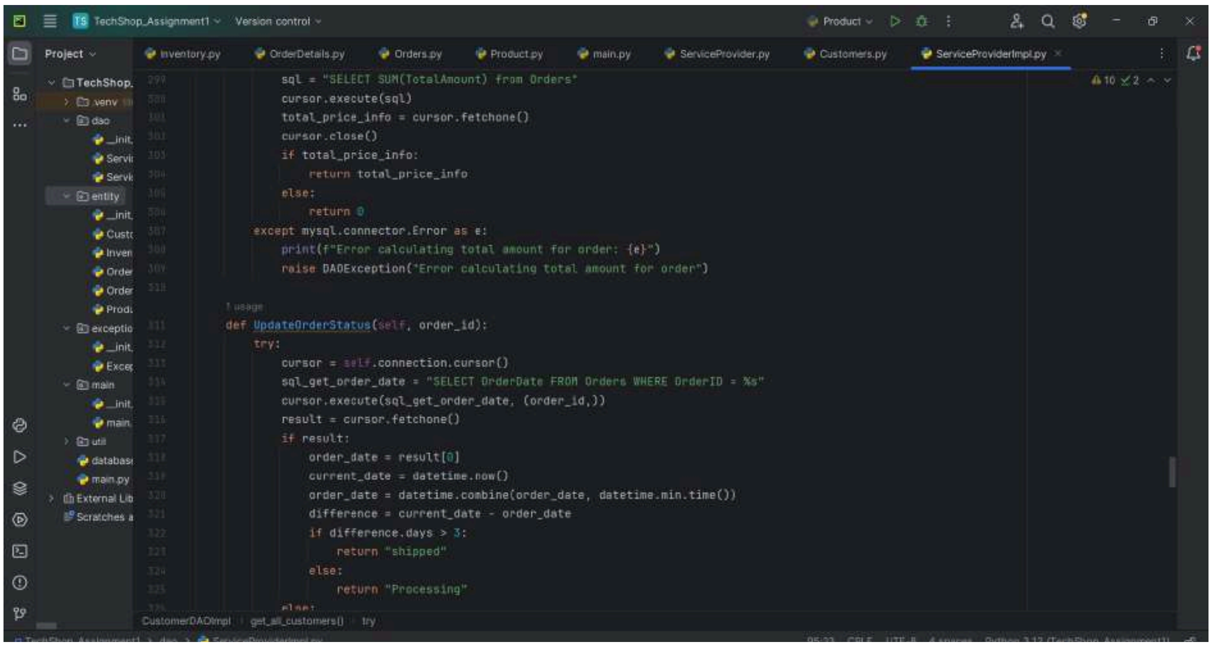
```
                total_amount += order_detail.Quantity * price
            sql_insert_order = "INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES (%s, %s, %s)"
            order_data = (order.CustomerID, order.orderDate, total_amount)
            cursor.execute(sql_insert_order, order_data)
            order_id = cursor.lastrowid
            for order_detail in order_details:
                sql_insert_order_detail = "INSERT INTO OrderDetails (OrderID, ProductID, Quantity) VALUES (%s, %s, %s)"
                order_detail_data = (order_id, order_detail.ProductID, order_detail.Quantity)
                cursor.execute(sql_insert_order_detail, order_detail_data)
            self.connection.commit()
            return True
        except mysql.connector.Error as e:
            print(f"Error creating customer: {e}")
            raise DAOException("Error adding order")

    1 usage
    def display_orders(self):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT * FROM Orders"
            cursor.execute(sql)
            results = cursor.fetchall()
            return results
        except mysql.connector.Error as e:
            print(f"Error fetching all customers: {e}")
            raise DAOException("Error fetching all customers")

    1 usage
    def CancelOrder(self, order_id):
```



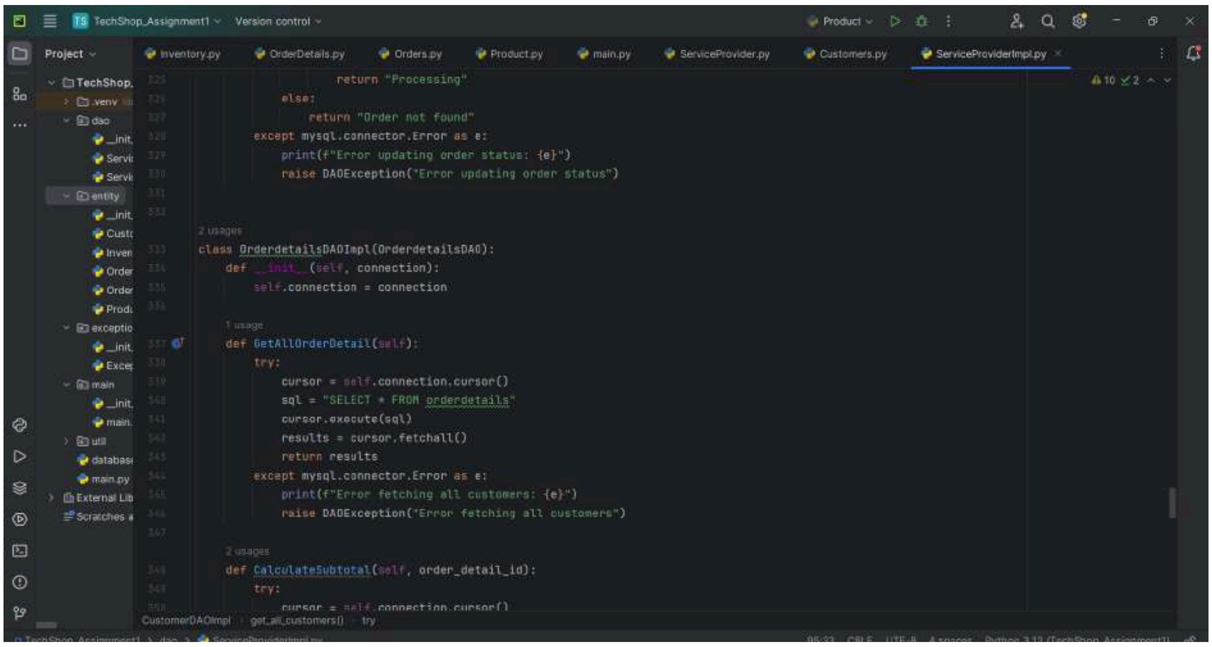
```
    1 usage
    def CancelOrder(self, order_id):
        try:
            cursor = self.connection.cursor()
            sql_select_order = "SELECT OrderID FROM Orders WHERE OrderID = %s"
            cursor.execute(sql_select_order, (order_id,))
            result = cursor.fetchone()
            if not result:
                raise DAOException("Order not found")
            sql_select_order_details = "SELECT ProductID, Quantity FROM OrderDetails WHERE OrderID = %s"
            cursor.execute(sql_select_order_details, (order_id,))
            order_details = cursor.fetchall()
            for product_id, quantity in order_details:
                sql_update_inventory = ("UPDATE Inventory SET QuantityInStock ="
                                        " QuantityInStock + %s WHERE ProductID = %s")
                cursor.execute(sql_update_inventory, (quantity, product_id))
            sql_delete_order_details = "DELETE FROM OrderDetails WHERE OrderID = %s"
            cursor.execute(sql_delete_order_details, (order_id,))
            sql_delete_order = "DELETE FROM Orders WHERE OrderID = %s"
            cursor.execute(sql_delete_order, (order_id,))
            self.connection.commit()
            cursor.close()
            return True
        except mysql.connector.Error as e:
            print(f"Error deleting product: {e}")
            raise DAOException("Error deleting product")

    2 usages
    def GetOrderDetails(self, order_id):
```



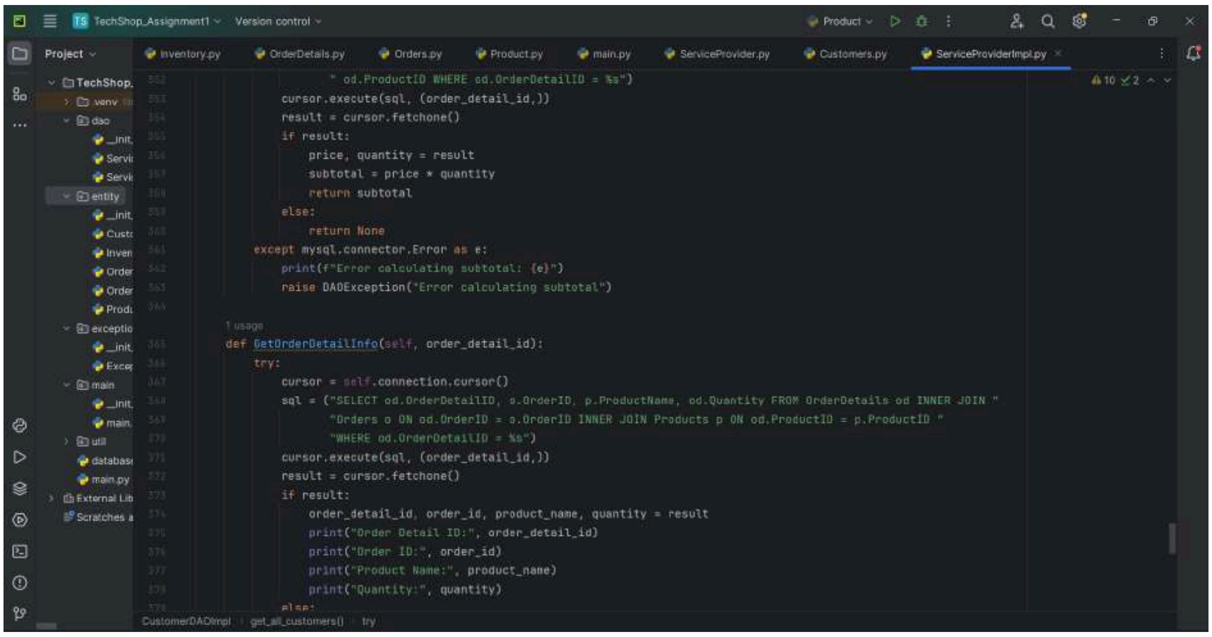
```python
    2 usages
    def GetOrderDetails(self, order_id):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT * FROM Orders WHERE OrderID = %s"
            cursor.execute(sql, (order_id,))
            results = cursor.fetchall()
            cursor.close()
            orders = []
            if results:
                for result in results:
                    order = Order(result[0], result[1], result[2], result[3])
                    orders.append(order)
                return orders
            else:
                return None
        except mysql.connector.Error as e:
            print(f"Error fetching customer: {e}")
            raise DAOException("Error fetching customer")

    1 usage
    def CalculateTotalAmount(self):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT SUM(TotalAmount) from Orders"
            cursor.execute(sql)
            total_price_info = cursor.fetchone()
            cursor.close()
            if total_price_info:
```



```python
            sql = "SELECT SUM(TotalAmount) from Orders"
            cursor.execute(sql)
            total_price_info = cursor.fetchone()
            cursor.close()
            if total_price_info:
                return total_price_info
            else:
                return 0
        except mysql.connector.Error as e:
            print(f"Error calculating total amount for order: {e}")
            raise DAOException("Error calculating total amount for order")

    1 usage
    def UpdateOrderStatus(self, order_id):
        try:
            cursor = self.connection.cursor()
            sql_get_order_date = "SELECT OrderDate FROM Orders WHERE OrderID = %s"
            cursor.execute(sql_get_order_date, (order_id,))
            result = cursor.fetchone()
            if result:
                order_date = result[0]
                current_date = datetime.now()
                order_date = datetime.combine(order_date, datetime.min.time())
                difference = current_date - order_date
                if difference.days > 3:
                    return "shipped"
                else:
                    return "Processing"
            else:
```



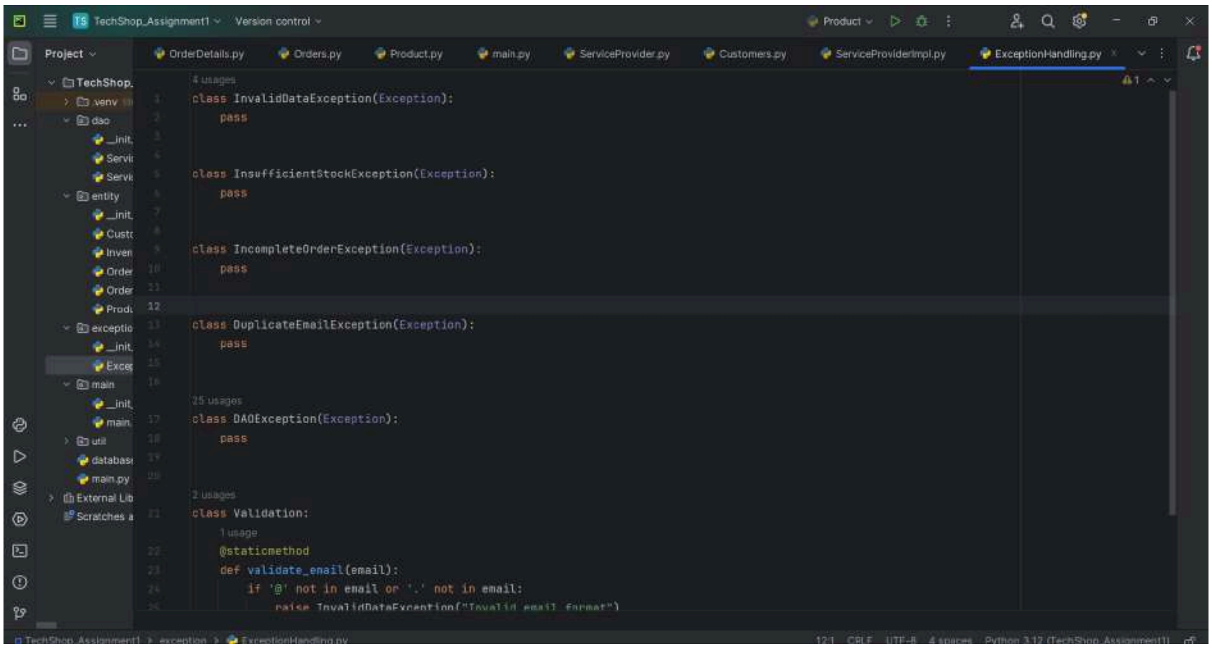
```
                    return "Processing"
            else:
                return "Order not found"
        except mysql.connector.Error as e:
            print(f"Error updating order status: {e}")
            raise DAOException("Error updating order status")


2 usages
class OrderdetailsDAOImpl(OrderdetailsDAO):
    def __init__(self, connection):
        self.connection = connection

    1 usage
    def GetAllOrderDetail(self):
        try:
            cursor = self.connection.cursor()
            sql = "SELECT * FROM orderdetails"
            cursor.execute(sql)
            results = cursor.fetchall()
            return results
        except mysql.connector.Error as e:
            print(f"Error fetching all customers: {e}")
            raise DAOException("Error fetching all customers")

    2 usages
    def CalculateSubtotal(self, order_detail_id):
        try:
            cursor = self.connection.cursor()
```



```
                   " od.ProductID WHERE od.OrderDetailID = %s")
            cursor.execute(sql, (order_detail_id,))
            result = cursor.fetchone()
            if result:
                price, quantity = result
                subtotal = price * quantity
                return subtotal
            else:
                return None
        except mysql.connector.Error as e:
            print(f"Error calculating subtotal: {e}")
            raise DAOException("Error calculating subtotal")

    1 usage
    def GetOrderDetailInfo(self, order_detail_id):
        try:
            cursor = self.connection.cursor()
            sql = ("SELECT od.OrderDetailID, o.OrderID, p.ProductName, od.Quantity FROM OrderDetails od INNER JOIN "
                   "Orders o ON od.OrderID = o.OrderID INNER JOIN Products p ON od.ProductID = p.ProductID "
                   "WHERE od.OrderDetailID = %s")
            cursor.execute(sql, (order_detail_id,))
            result = cursor.fetchone()
            if result:
                order_detail_id, order_id, product_name, quantity = result
                print("Order Detail ID:", order_detail_id)
                print("Order ID:", order_id)
                print("Product Name:", product_name)
                print("Quantity:", quantity)
            else:
```

Task 5: Exceptions handling

• Data Validation:
• Inventory Management:
• Order Processing:
• Payment Processing:
• Database Access:
• Concurrency Control:
• Security and Authentication:

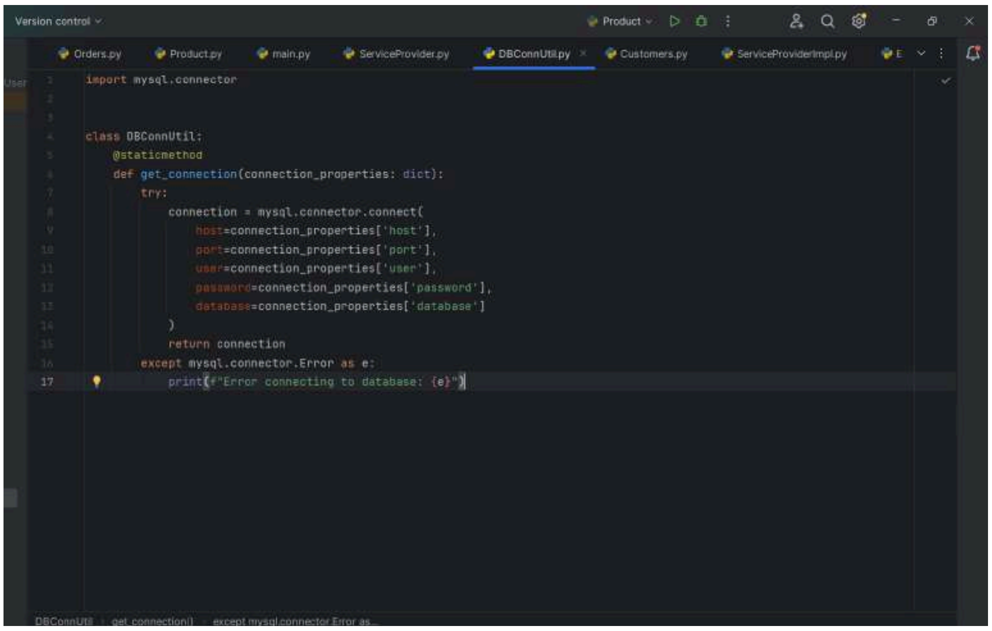


```python
class InvalidDataException(Exception):
    pass


class InsufficientStockException(Exception):
    pass


class IncompleteOrderException(Exception):
    pass


class DuplicateEmailException(Exception):
    pass


class DAOException(Exception):
    pass


class Validation:
    @staticmethod
    def validate_email(email):
        if '@' not in email or '.' not in email:
            raise InvalidDataException("Invalid email format")
```

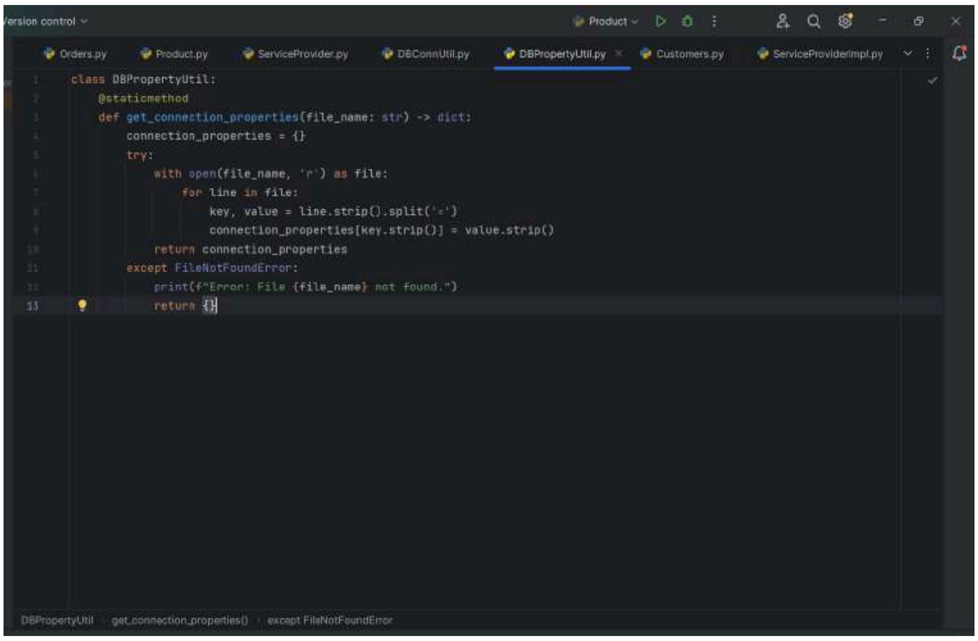
Task 7: Database Connectivity

• 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.

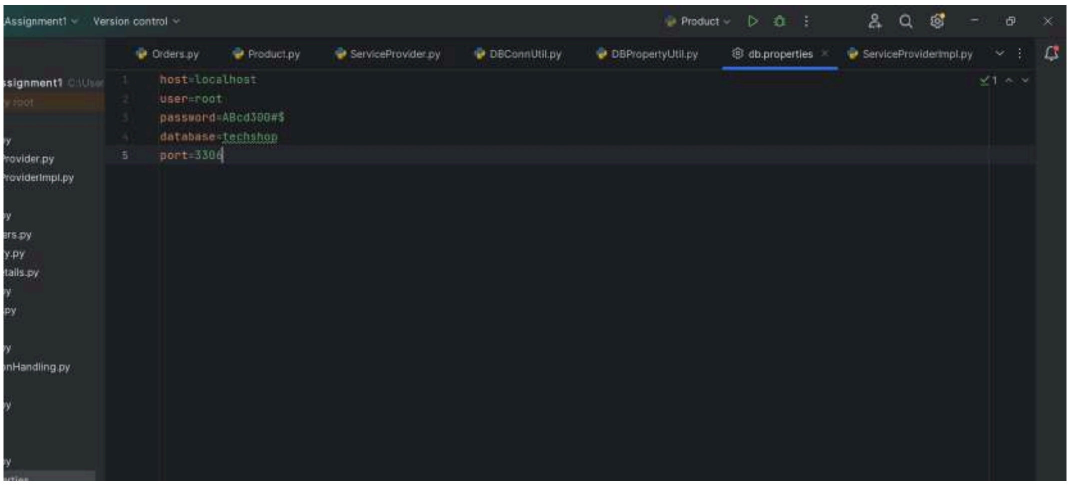
• Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.



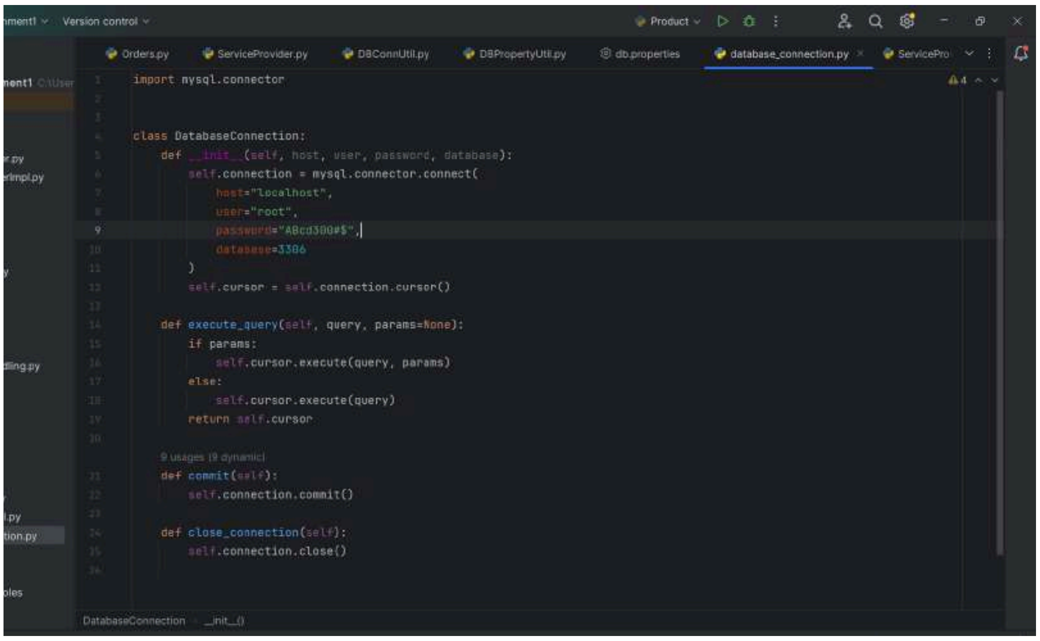
```python
import mysql.connector


class DBConnUtil:
    @staticmethod
    def get_connection(connection_properties: dict):
        try:
            connection = mysql.connector.connect(
                host=connection_properties['host'],
                port=connection_properties['port'],
                user=connection_properties['user'],
                password=connection_properties['password'],
                database=connection_properties['database']
            )
            return connection
        except mysql.connector.Error as e:
            print(f"Error connecting to database: {e}")
```



```python
class DBPropertyUtil:
    @staticmethod
    def get_connection_properties(file_name: str) -> dict:
        connection_properties = {}
        try:
            with open(file_name, 'r') as file:
                for line in file:
                    key, value = line.strip().split('=')
                    connection_properties[key.strip()] = value.strip()
            return connection_properties
        except FileNotFoundError:
            print(f"Error: File {file_name} not found.")
            return {}
```

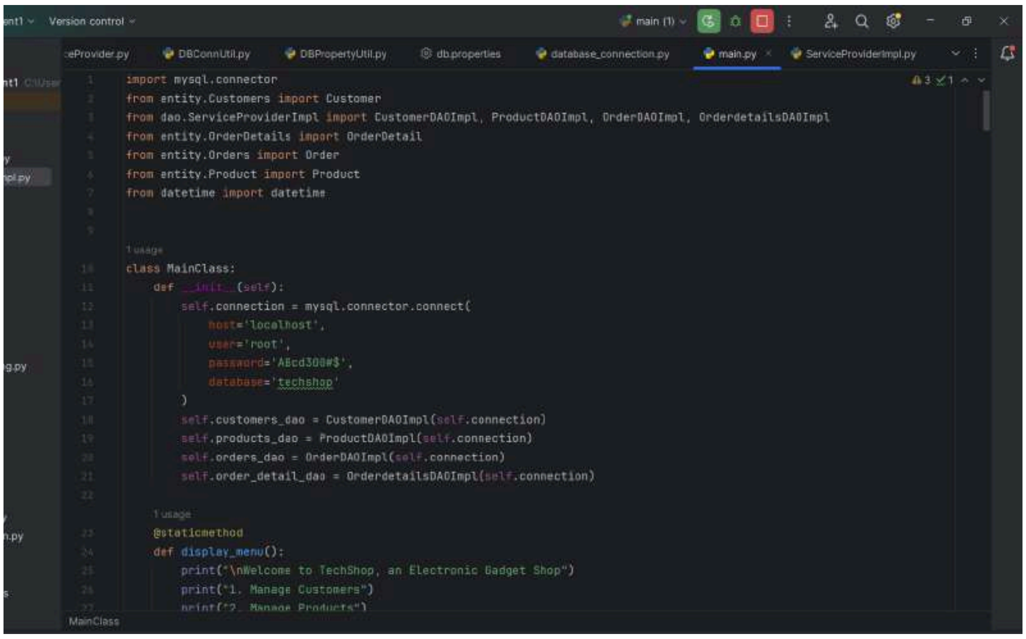


```
host=localhost
user=root
password=ABcd300#$
database=techshop
port=3306
```

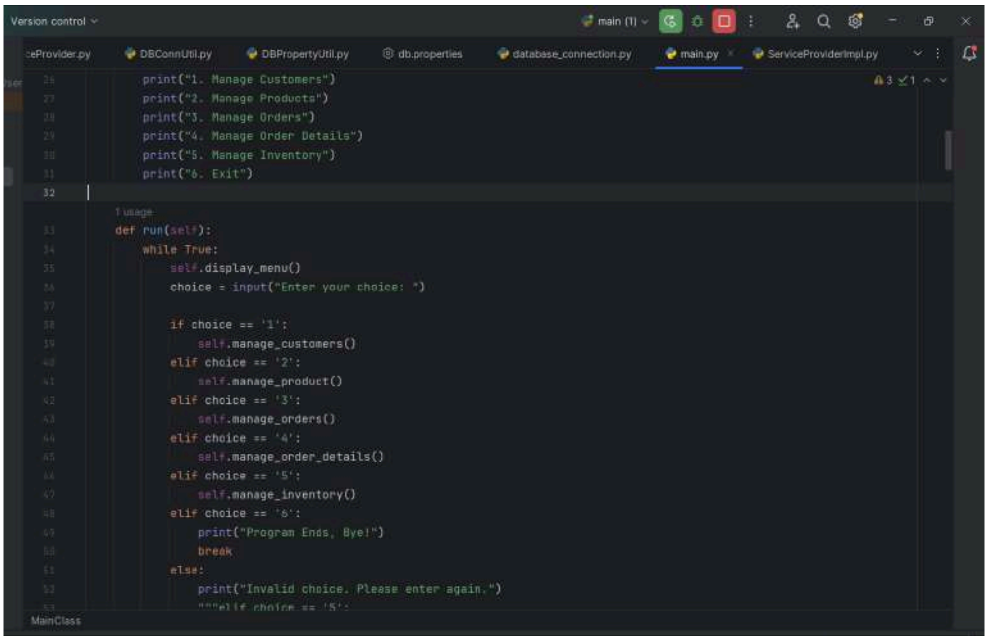


```python
import mysql.connector


class DatabaseConnection:
    def __init__(self, host, user, password, database):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="ABcd300#$",
            database=3306
        )
        self.cursor = self.connection.cursor()

    def execute_query(self, query, params=None):
        if params:
            self.cursor.execute(query, params)
        else:
            self.cursor.execute(query)
        return self.cursor

    def commit(self):
        self.connection.commit()

    def close_connection(self):
        self.connection.close()
```

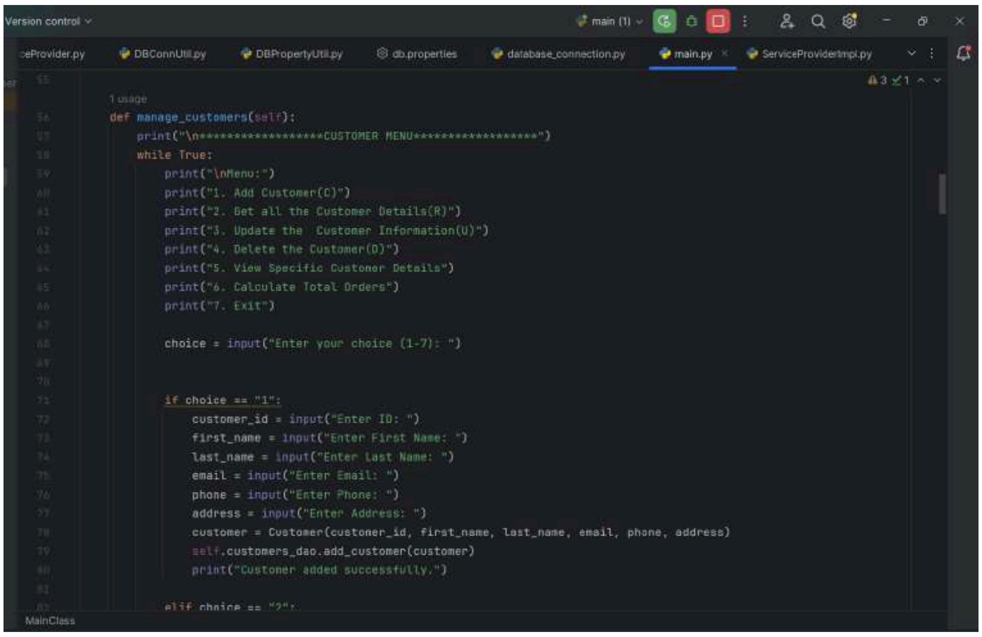
1: Customer Registration
2: Product Catalog Management’
3: Placing Customer Orders
4: Tracking Order Status
6: Sales Reporting
7: Customer Account Updates
8: Payment Processing
9: Product Search and Recommendations



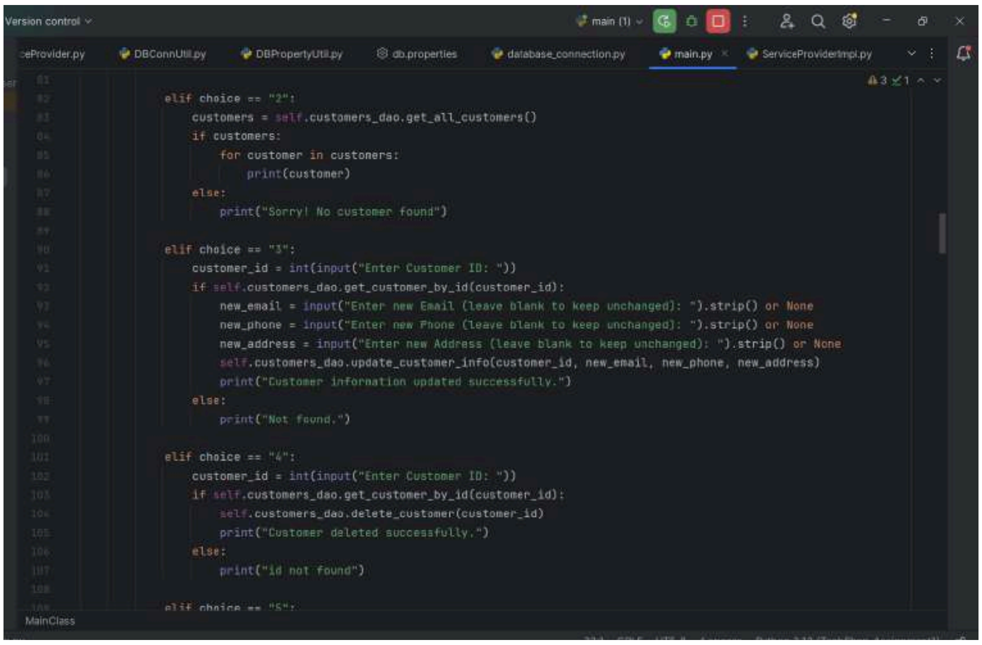
```python
import mysql.connector
from entity.Customers import Customer
from dao.ServiceProviderImpl import CustomerDAOImpl, ProductDAOImpl, OrderDAOImpl, OrderdetailsDAOImpl
from entity.OrderDetails import OrderDetail
from entity.Orders import Order
from entity.Product import Product
from datetime import datetime


class MainClass:
    def __init__(self):
        self.connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='ABcd300#$',
            database='techshop'
        )
        self.customers_dao = CustomerDAOImpl(self.connection)
        self.products_dao = ProductDAOImpl(self.connection)
        self.orders_dao = OrderDAOImpl(self.connection)
        self.order_detail_dao = OrderdetailsDAOImpl(self.connection)

    @staticmethod
    def display_menu():
        print("\nWelcome to TechShop, an Electronic Gadget Shop")
        print("1. Manage Customers")
        print("2. Manage Products")
```



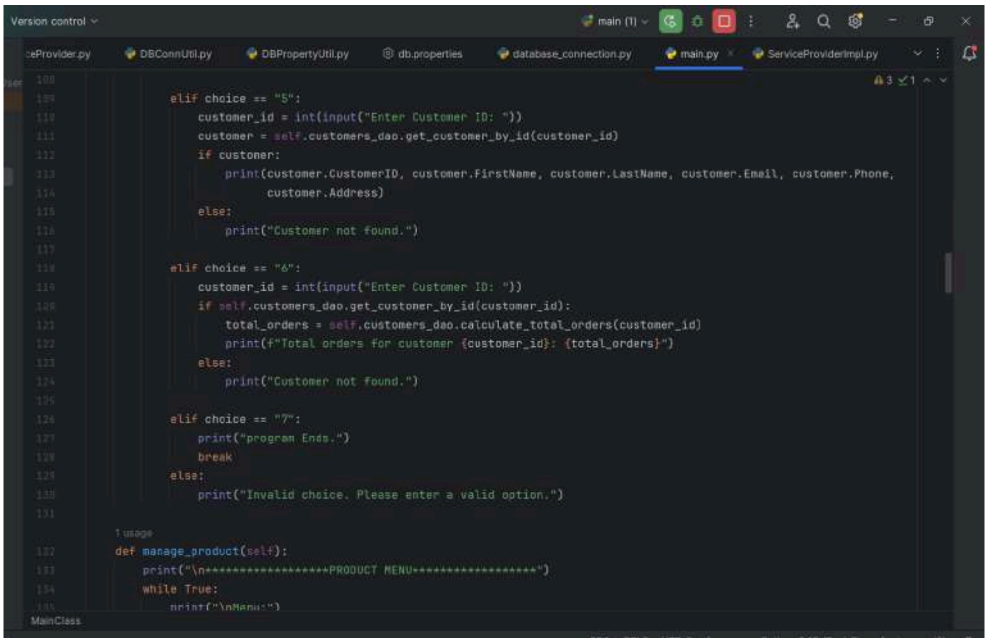
```python
        print("1. Manage Customers")
        print("2. Manage Products")
        print("3. Manage Orders")
        print("4. Manage Order Details")
        print("5. Manage Inventory")
        print("6. Exit")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")

            if choice == '1':
                self.manage_customers()
            elif choice == '2':
                self.manage_product()
            elif choice == '3':
                self.manage_orders()
            elif choice == '4':
                self.manage_order_details()
            elif choice == '5':
                self.manage_inventory()
            elif choice == '6':
                print("Program Ends, Bye!")
                break
            else:
                print("Invalid choice. Please enter again.")
                """elif choice == '5':
```



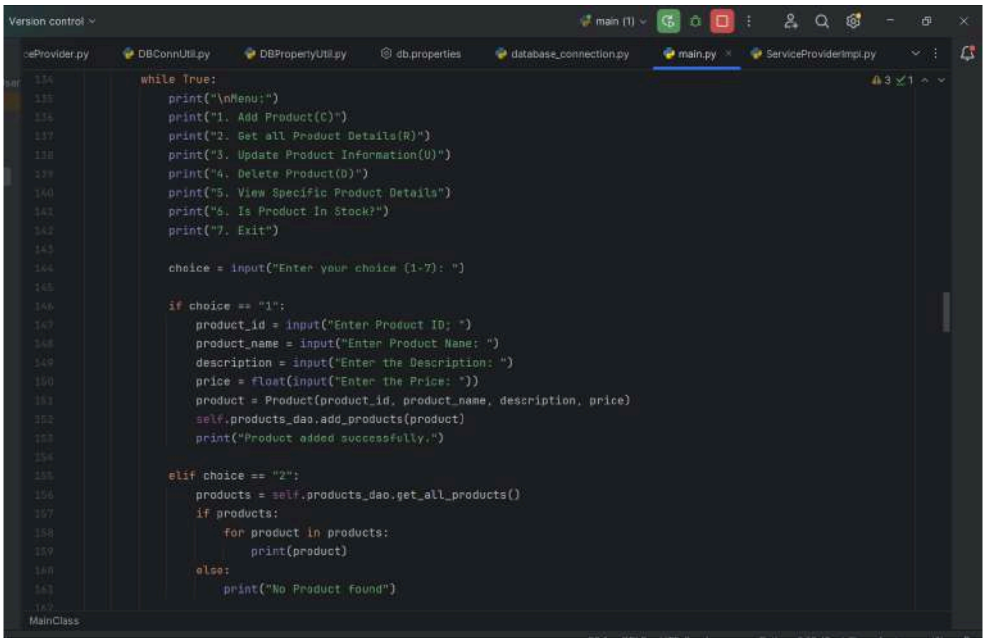
```python
    1 usage
    def manage_customers(self):
        print("\n******************CUSTOMER MENU******************")
        while True:
            print("\nMenu:")
            print("1. Add Customer(C)")
            print("2. Get all the Customer Details(R)")
            print("3. Update the  Customer Information(U)")
            print("4. Delete the Customer(D)")
            print("5. View Specific Customer Details")
            print("6. Calculate Total Orders")
            print("7. Exit")

            choice = input("Enter your choice (1-7): ")


            if choice == "1":
                customer_id = input("Enter ID: ")
                first_name = input("Enter First Name: ")
                last_name = input("Enter Last Name: ")
                email = input("Enter Email: ")
                phone = input("Enter Phone: ")
                address = input("Enter Address: ")
                customer = Customer(customer_id, first_name, last_name, email, phone, address)
                self.customers_dao.add_customer(customer)
                print("Customer added successfully.")

            elif choice == "2":
```



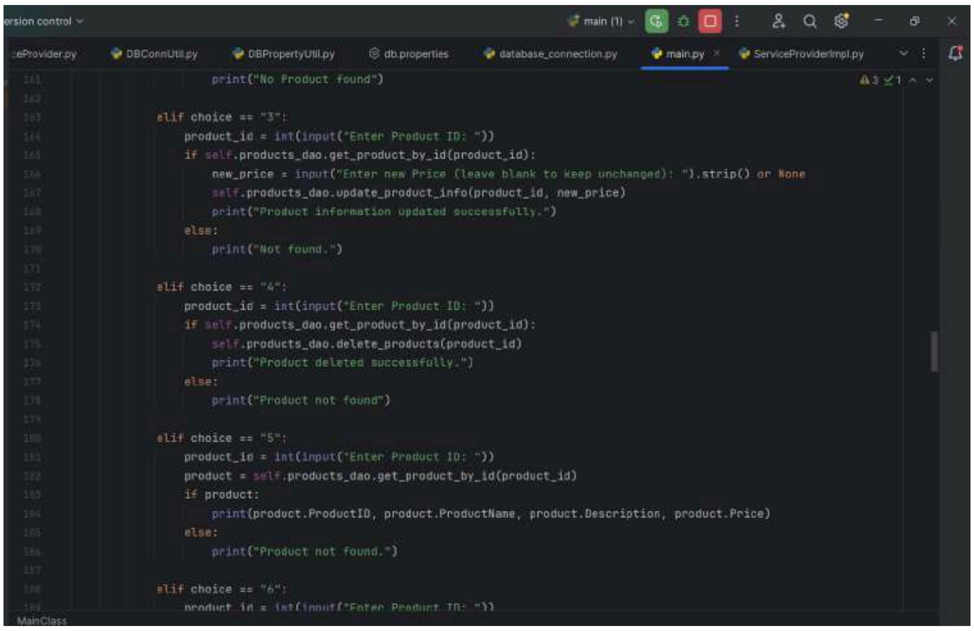
```python
            elif choice == "2":
                customers = self.customers_dao.get_all_customers()
                if customers:
                    for customer in customers:
                        print(customer)
                else:
                    print("Sorry! No customer found")

            elif choice == "3":
                customer_id = int(input("Enter Customer ID: "))
                if self.customers_dao.get_customer_by_id(customer_id):
                    new_email = input("Enter new Email (leave blank to keep unchanged): ").strip() or None
                    new_phone = input("Enter new Phone (leave blank to keep unchanged): ").strip() or None
                    new_address = input("Enter new Address (leave blank to keep unchanged): ").strip() or None
                    self.customers_dao.update_customer_info(customer_id, new_email, new_phone, new_address)
                    print("Customer information updated successfully.")
                else:
                    print("Not found.")

            elif choice == "4":
                customer_id = int(input("Enter Customer ID: "))
                if self.customers_dao.get_customer_by_id(customer_id):
                    self.customers_dao.delete_customer(customer_id)
                    print("Customer deleted successfully.")
                else:
                    print("id not found")

            elif choice == "5":
```



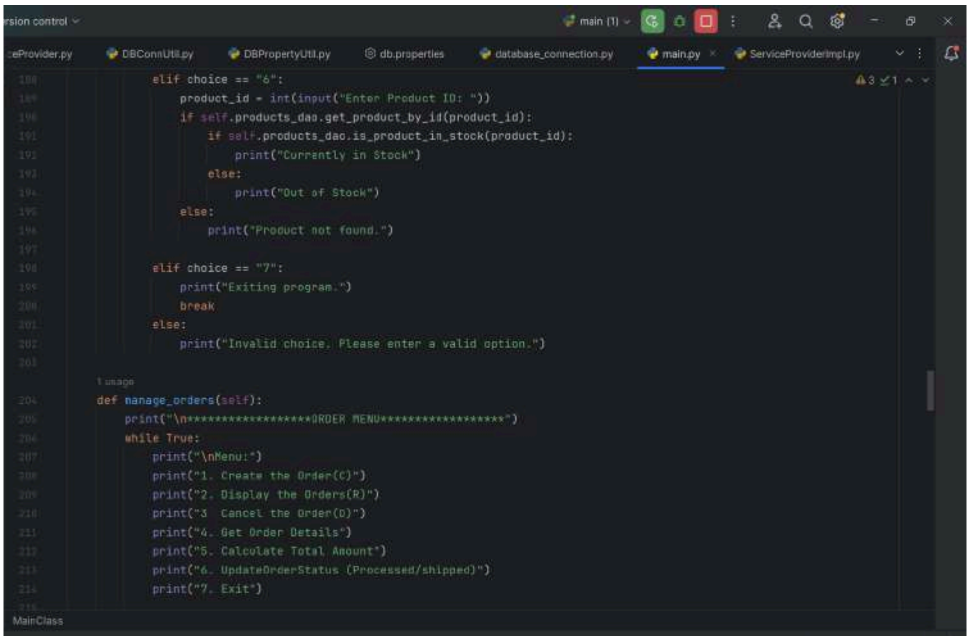
```python
            elif choice == "5":
                customer_id = int(input("Enter Customer ID: "))
                customer = self.customers_dao.get_customer_by_id(customer_id)
                if customer:
                    print(customer.CustomerID, customer.FirstName, customer.LastName, customer.Email, customer.Phone,
                          customer.Address)
                else:
                    print("Customer not found.")

            elif choice == "6":
                customer_id = int(input("Enter Customer ID: "))
                if self.customers_dao.get_customer_by_id(customer_id):
                    total_orders = self.customers_dao.calculate_total_orders(customer_id)
                    print(f"Total orders for customer {customer_id}: {total_orders}")
                else:
                    print("Customer not found.")

            elif choice == "7":
                print("program Ends.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")

    def manage_product(self):
        print("\n******************PRODUCT MENU******************")
        while True:
```



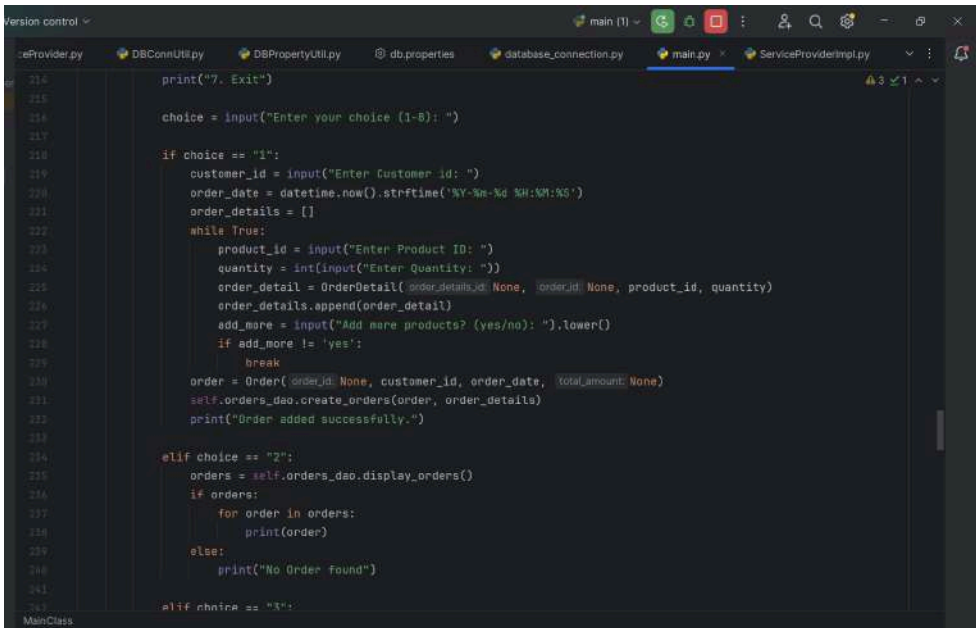
```python
        while True:
            print("\nMenu:")
            print("1. Add Product(C)")
            print("2. Get all Product Details(R)")
            print("3. Update Product Information(U)")
            print("4. Delete Product(D)")
            print("5. View Specific Product Details")
            print("6. Is Product In Stock?")
            print("7. Exit")

            choice = input("Enter your choice (1-7): ")

            if choice == "1":
                product_id = input("Enter Product ID: ")
                product_name = input("Enter Product Name: ")
                description = input("Enter the Description: ")
                price = float(input("Enter the Price: "))
                product = Product(product_id, product_name, description, price)
                self.products_dao.add_products(product)
                print("Product added successfully.")

            elif choice == "2":
                products = self.products_dao.get_all_products()
                if products:
                    for product in products:
                        print(product)
                else:
                    print("No Product found")
```



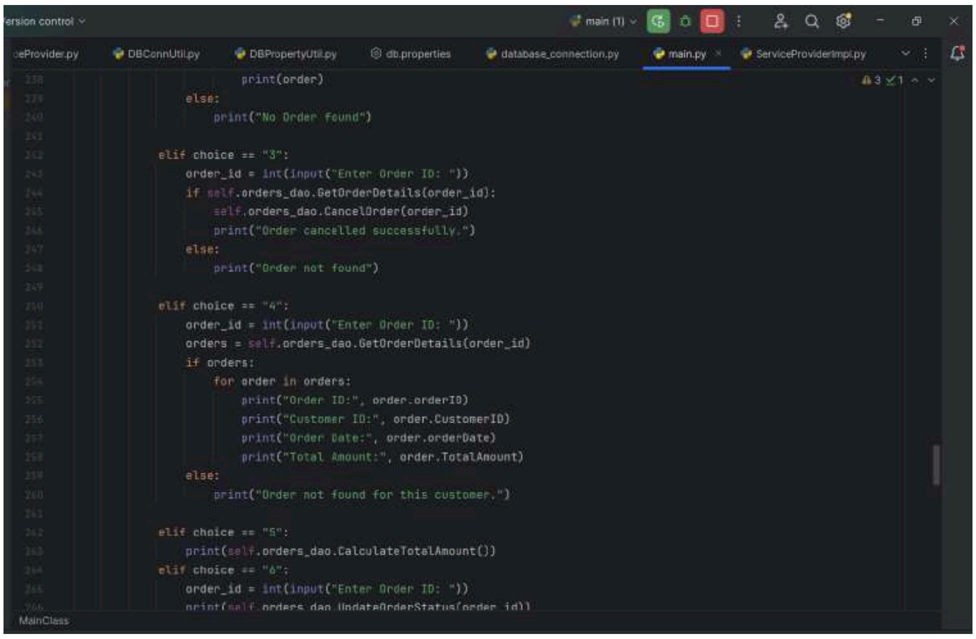
```python
                    print("No Product found")

            elif choice == "3":
                product_id = int(input("Enter Product ID: "))
                if self.products_dao.get_product_by_id(product_id):
                    new_price = input("Enter new Price (leave blank to keep unchanged): ").strip() or None
                    self.products_dao.update_product_info(product_id, new_price)
                    print("Product information updated successfully.")
                else:
                    print("Not found.")

            elif choice == "4":
                product_id = int(input("Enter Product ID: "))
                if self.products_dao.get_product_by_id(product_id):
                    self.products_dao.delete_products(product_id)
                    print("Product deleted successfully.")
                else:
                    print("Product not found")

            elif choice == "5":
                product_id = int(input("Enter Product ID: "))
                product = self.products_dao.get_product_by_id(product_id)
                if product:
                    print(product.ProductID, product.ProductName, product.Description, product.Price)
                else:
                    print("Product not found.")

            elif choice == "6":
                product_id = int(input("Enter Product ID: "))
```



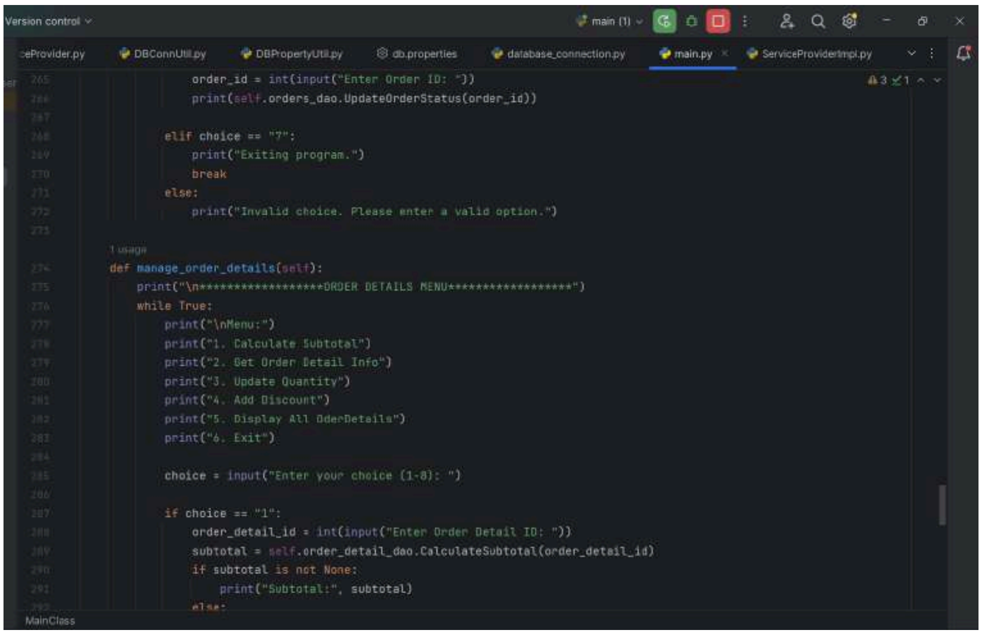
```python
            elif choice == "6":
                product_id = int(input("Enter Product ID: "))
                if self.products_dao.get_product_by_id(product_id):
                    if self.products_dao.is_product_in_stock(product_id):
                        print("Currently in Stock")
                    else:
                        print("Out of Stock")
                else:
                    print("Product not found.")

            elif choice == "7":
                print("Exiting program.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")

    1 usage
    def manage_orders(self):
        print("\n******************ORDER MENU******************")
        while True:
            print("\nMenu:")
            print("1. Create the Order(C)")
            print("2. Display the Orders(R)")
            print("3  Cancel the Order(D)")
            print("4. Get Order Details")
            print("5. Calculate Total Amount")
            print("6. UpdateOrderStatus (Processed/shipped)")
            print("7. Exit")
```



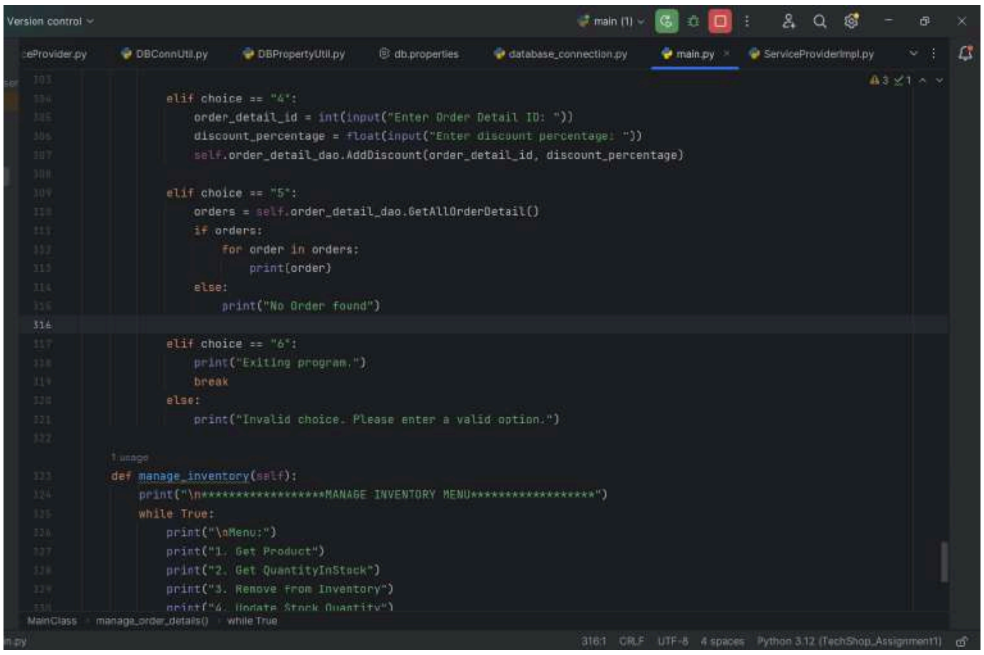
```python
            print("7. Exit")

            choice = input("Enter your choice (1-8): ")

            if choice == "1":
                customer_id = input("Enter Customer id: ")
                order_date = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                order_details = []
                while True:
                    product_id = input("Enter Product ID: ")
                    quantity = int(input("Enter Quantity: "))
                    order_detail = OrderDetail( order_details_id: None,  order_id: None, product_id, quantity)
                    order_details.append(order_detail)
                    add_more = input("Add more products? (yes/no): ").lower()
                    if add_more != 'yes':
                        break
                order = Order( order_id: None, customer_id, order_date,  total_amount: None)
                self.orders_dao.create_orders(order, order_details)
                print("Order added successfully.")

            elif choice == "2":
                orders = self.orders_dao.display_orders()
                if orders:
                    for order in orders:
                        print(order)
                else:
                    print("No Order found")

            elif choice == "3":
```



```python
                        print(order)
                else:
                    print("No Order found")

            elif choice == "3":
                order_id = int(input("Enter Order ID: "))
                if self.orders_dao.GetOrderDetails(order_id):
                    self.orders_dao.CancelOrder(order_id)
                    print("Order cancelled successfully.")
                else:
                    print("Order not found")

            elif choice == "4":
                order_id = int(input("Enter Order ID: "))
                orders = self.orders_dao.GetOrderDetails(order_id)
                if orders:
                    for order in orders:
                        print("Order ID:", order.orderID)
                        print("Customer ID:", order.CustomerID)
                        print("Order Date:", order.orderDate)
                        print("Total Amount:", order.TotalAmount)
                else:
                    print("Order not found for this customer.")

            elif choice == "5":
                print(self.orders_dao.CalculateTotalAmount())
            elif choice == "6":
                order_id = int(input("Enter Order ID: "))
                print(self.orders_dao.UpdateOrderStatus(order_id))
```

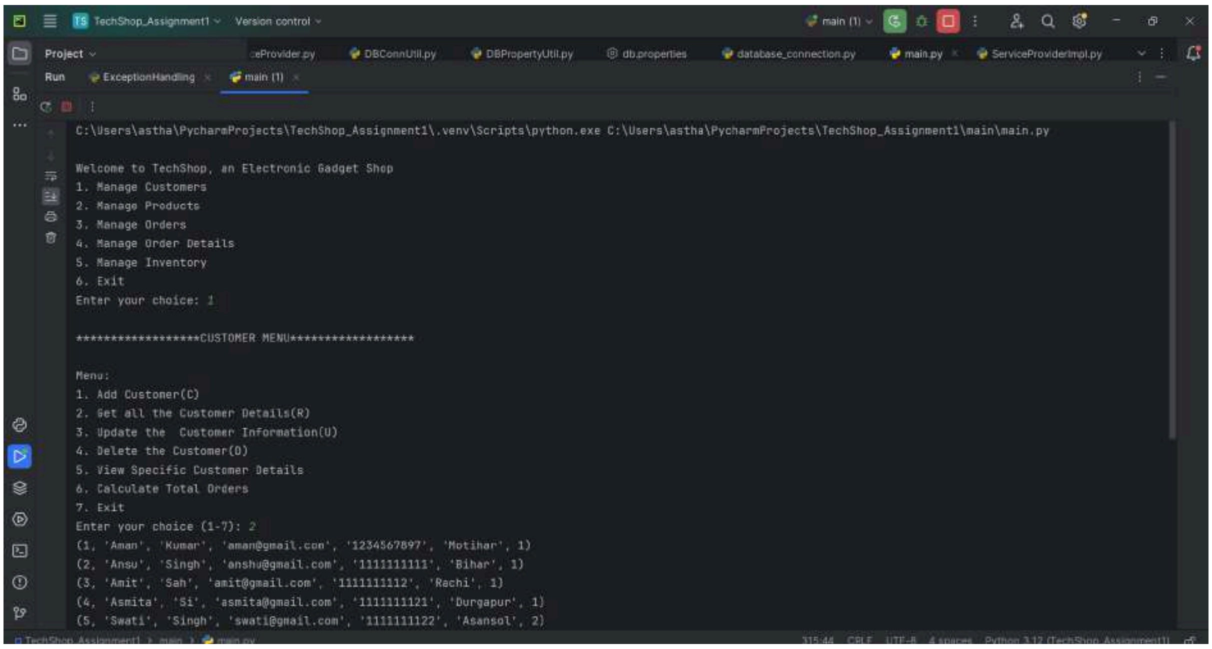


```python
                order_id = int(input("Enter Order ID: "))
                print(self.orders_dao.UpdateOrderStatus(order_id))

            elif choice == "7":
                print("Exiting program.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")

    def manage_order_details(self):
        print("\n******************ORDER DETAILS MENU******************")
        while True:
            print("\nMenu:")
            print("1. Calculate Subtotal")
            print("2. Get Order Detail Info")
            print("3. Update Quantity")
            print("4. Add Discount")
            print("5. Display All OderDetails")
            print("6. Exit")

            choice = input("Enter your choice (1-8): ")

            if choice == "1":
                order_detail_id = int(input("Enter Order Detail ID: "))
                subtotal = self.order_detail_dao.CalculateSubtotal(order_detail_id)
                if subtotal is not None:
                    print("Subtotal:", subtotal)
                else:
```

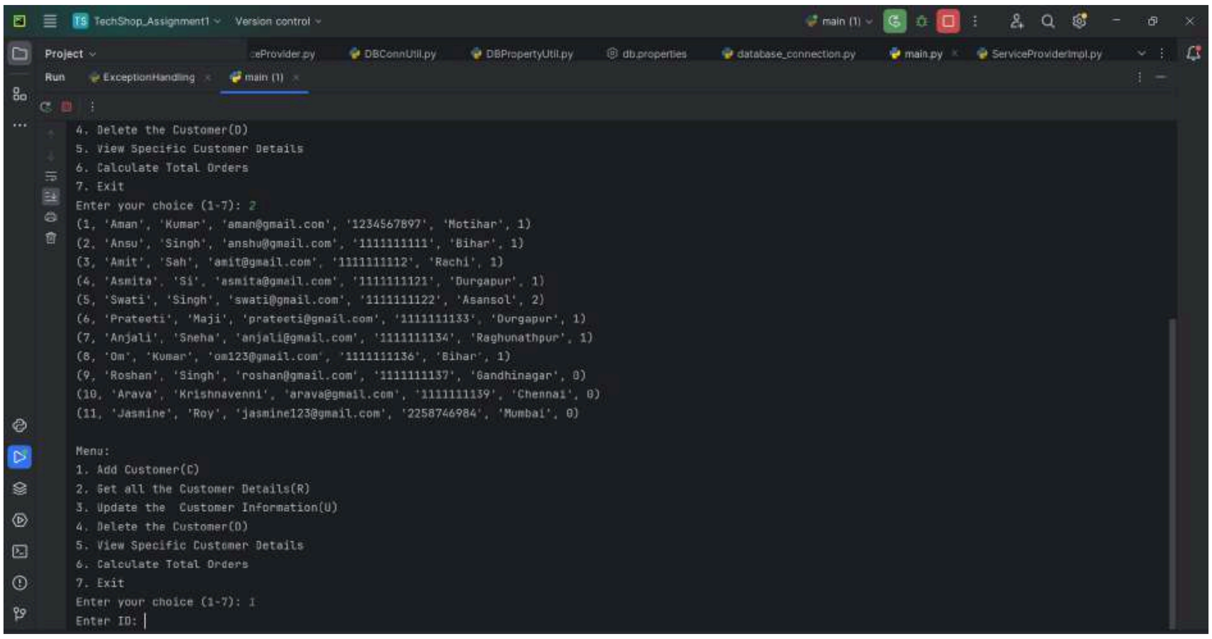


```python

            elif choice == "4":
                order_detail_id = int(input("Enter Order Detail ID: "))
                discount_percentage = float(input("Enter discount percentage: "))
                self.order_detail_dao.AddDiscount(order_detail_id, discount_percentage)

            elif choice == "5":
                orders = self.order_detail_dao.GetAllOrderDetail()
                if orders:
                    for order in orders:
                        print(order)
                else:
                    print("No Order found")

            elif choice == "6":
                print("Exiting program.")
                break
            else:
                print("Invalid choice. Please enter a valid option.")

    def manage_inventory(self):
        print("\n******************MANAGE INVENTORY MENU******************")
        while True:
            print("\nMenu:")
            print("1. Get Product")
            print("2. Get QuantityInStock")
            print("3. Remove from Inventory")
            print("4. Update Stock Quantity")
```

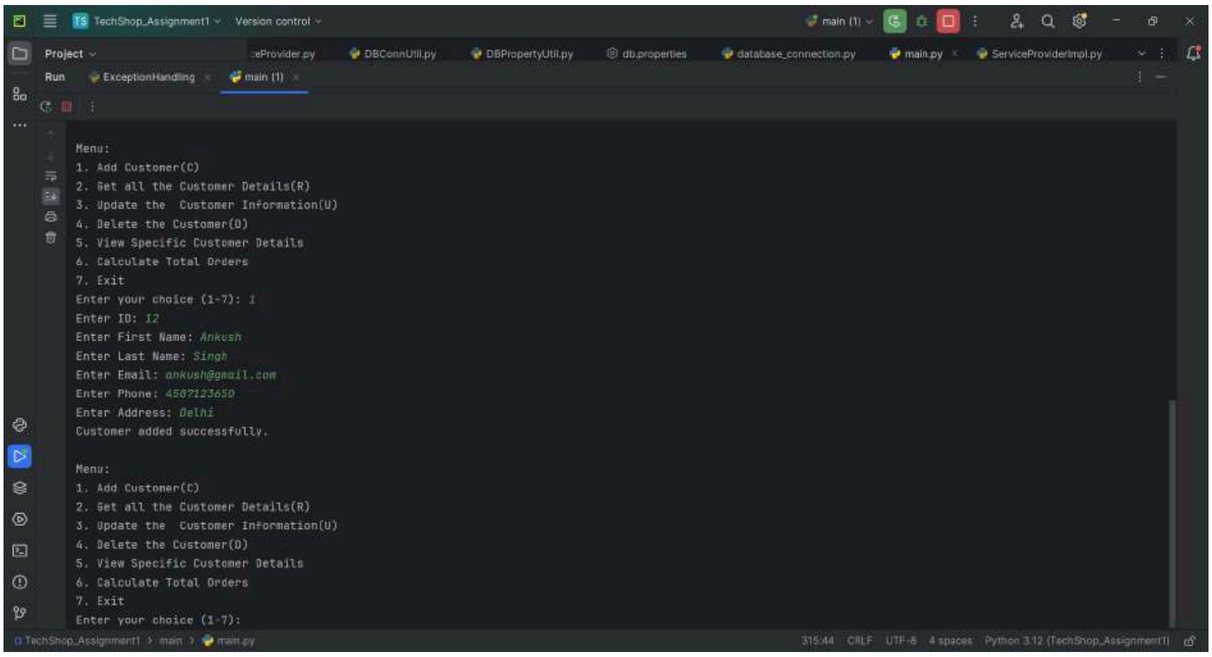
OUTPUTS



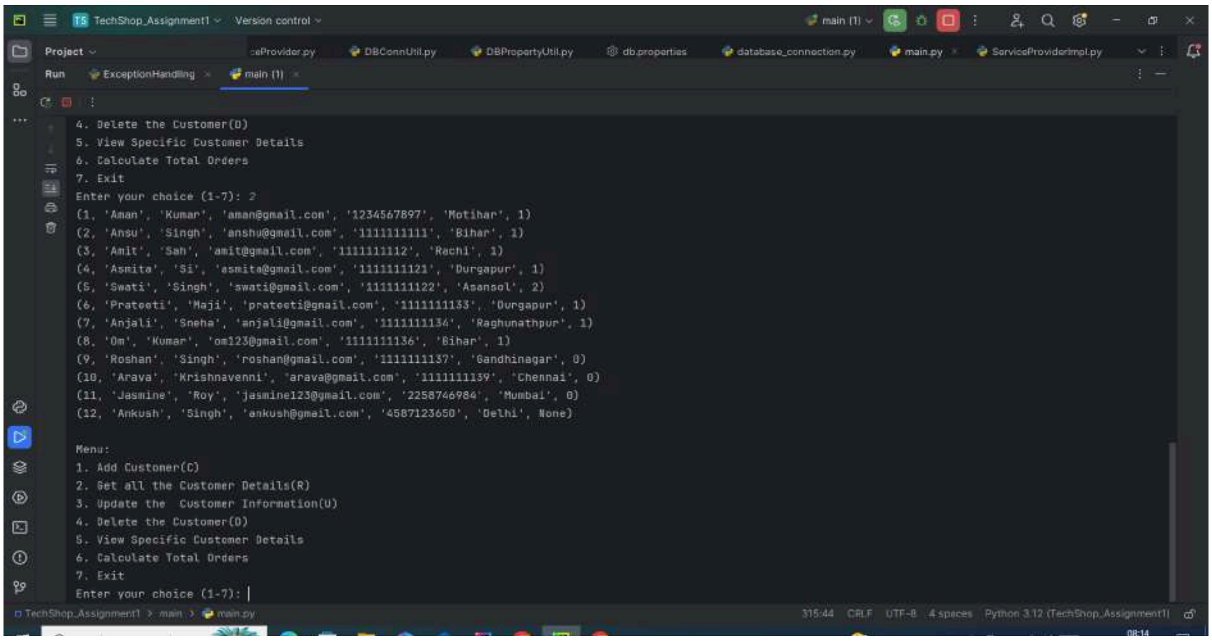
```
C:\Users\astha\PycharmProjects\TechShop_Assignment1\.venv\Scripts\python.exe C:\Users\astha\PycharmProjects\TechShop_Assignment1\main\main.py

Welcome to TechShop, an Electronic Gadget Shop
1. Manage Customers
2. Manage Products
3. Manage Orders
4. Manage Order Details
5. Manage Inventory
6. Exit
Enter your choice: 1

******************CUSTOMER MENU******************

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 2
(1, 'Aman', 'Kumar', 'aman@gmail.com', '1234567897', 'Motihar', 1)
(2, 'Ansu', 'Singh', 'anshu@gmail.com', '1111111111', 'Bihar', 1)
(3, 'Amit', 'Sah', 'amit@gmail.com', '1111111112', 'Rachi', 1)
(4, 'Asmita', 'Si', 'asmita@gmail.com', '1111111121', 'Durgapur', 1)
(5, 'Swati', 'Singh', 'swati@gmail.com', '1111111122', 'Asansol', 2)
```



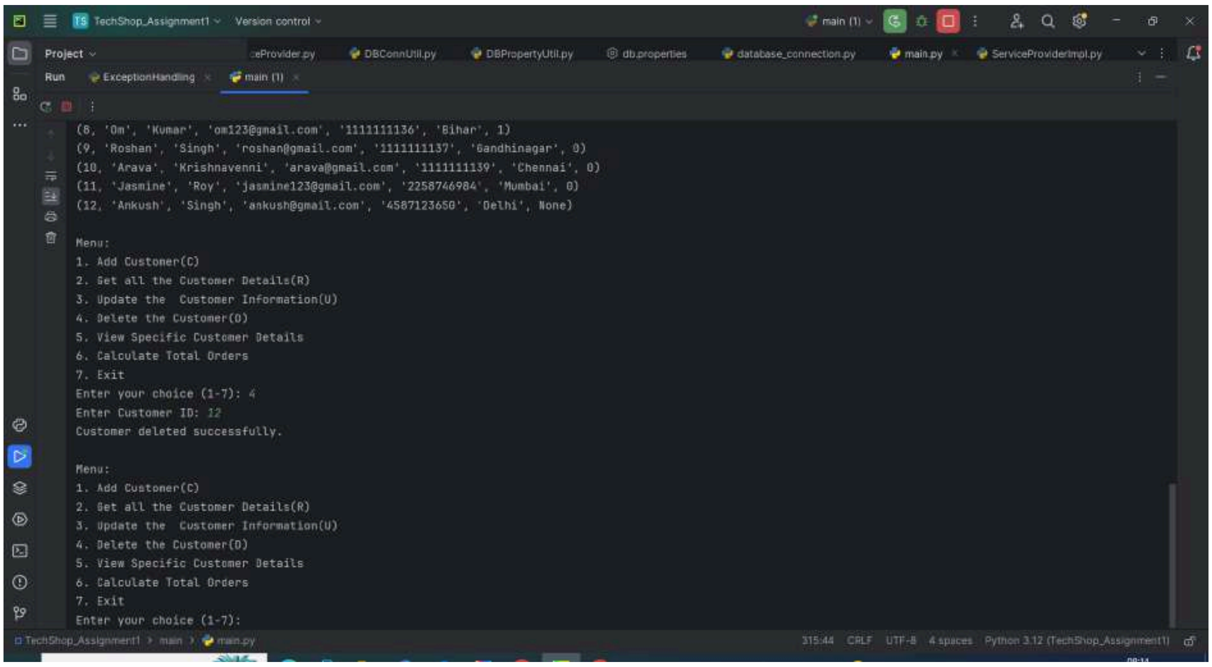
```
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 2
(1, 'Aman', 'Kumar', 'aman@gmail.com', '1234567897', 'Motihar', 1)
(2, 'Ansu', 'Singh', 'anshu@gmail.com', '1111111111', 'Bihar', 1)
(3, 'Amit', 'Sah', 'amit@gmail.com', '1111111112', 'Rachi', 1)
(4, 'Asmita', 'Si', 'asmita@gmail.com', '1111111121', 'Durgapur', 1)
(5, 'Swati', 'Singh', 'swati@gmail.com', '1111111122', 'Asansol', 2)
(6, 'Prateeti', 'Maji', 'prateeti@gmail.com', '1111111133', 'Durgapur', 1)
(7, 'Anjali', 'Sneha', 'anjali@gmail.com', '1111111134', 'Raghunathpur', 1)
(8, 'Om', 'Kumar', 'om123@gmail.com', '1111111136', 'Bihar', 1)
(9, 'Roshan', 'Singh', 'roshan@gmail.com', '1111111137', 'Gandhinagar', 0)
(10, 'Arava', 'Krishnavenni', 'arava@gmail.com', '1111111139', 'Chennai', 0)
(11, 'Jasmine', 'Roy', 'jasmine123@gmail.com', '2258746984', 'Mumbai', 0)

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 1
Enter ID: 
```



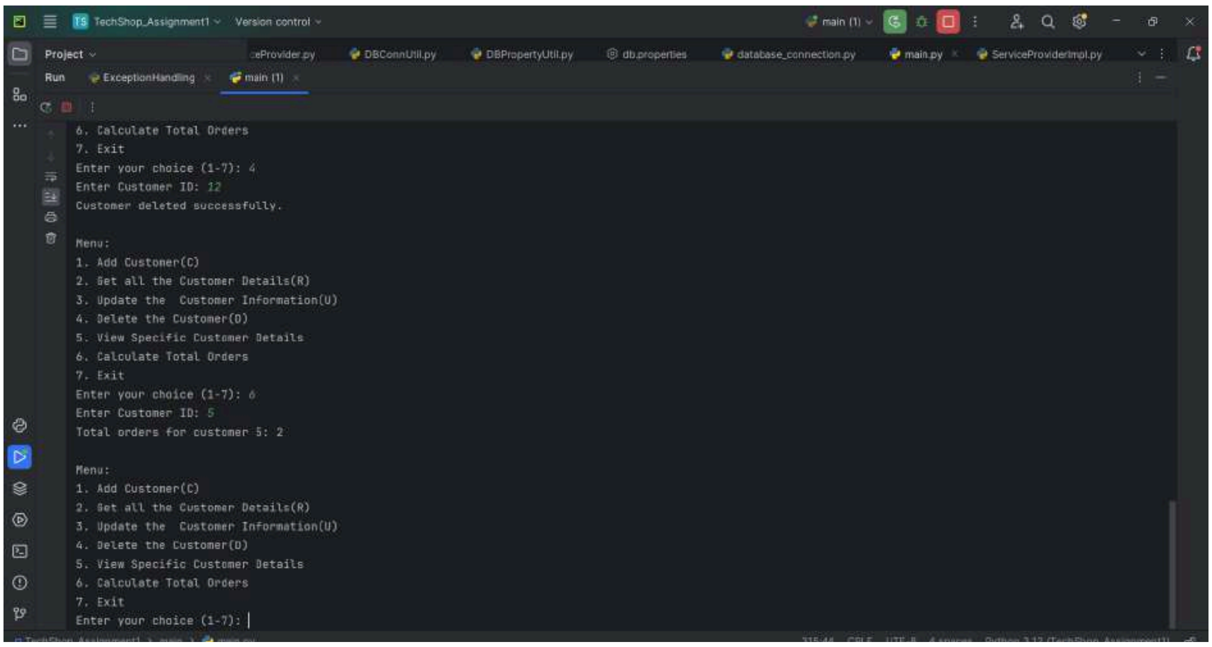
```
Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 1
Enter ID: 12
Enter First Name: Ankush
Enter Last Name: Singh
Enter Email: ankush@gmail.com
Enter Phone: 4587123650
Enter Address: Delhi
Customer added successfully.

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7):
```



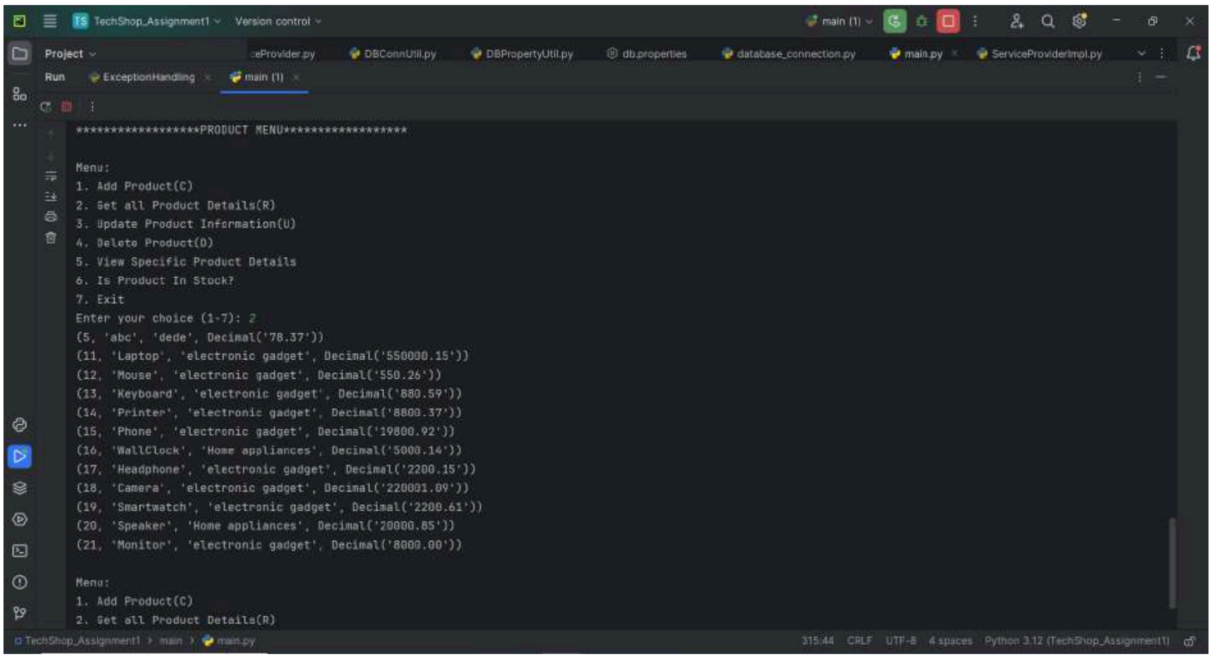
```
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 2
(1, 'Aman', 'Kumar', 'aman@gmail.com', '1234567897', 'Motihar', 1)
(2, 'Ansu', 'Singh', 'anshu@gmail.com', '1111111111', 'Bihar', 1)
(3, 'Amit', 'Sah', 'amit@gmail.com', '1111111112', 'Rachi', 1)
(4, 'Asmita', 'Si', 'asmita@gmail.com', '1111111121', 'Durgapur', 1)
(5, 'Swati', 'Singh', 'swati@gmail.com', '1111111122', 'Asansol', 2)
(6, 'Prateeti', 'Maji', 'prateeti@gmail.com', '1111111133', 'Durgapur', 1)
(7, 'Anjali', 'Sneha', 'anjali@gmail.com', '1111111134', 'Raghunathpur', 1)
(8, 'Om', 'Kumar', 'om123@gmail.com', '1111111136', 'Bihar', 1)
(9, 'Roshan', 'Singh', 'roshan@gmail.com', '1111111137', 'Gandhinagar', 0)
(10, 'Arava', 'Krishnavenni', 'arava@gmail.com', '1111111139', 'Chennai', 0)
(11, 'Jasmine', 'Roy', 'jasmine123@gmail.com', '2258746984', 'Mumbai', 0)
(12, 'Ankush', 'Singh', 'ankush@gmail.com', '4587123650', 'Delhi', None)

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7):
```



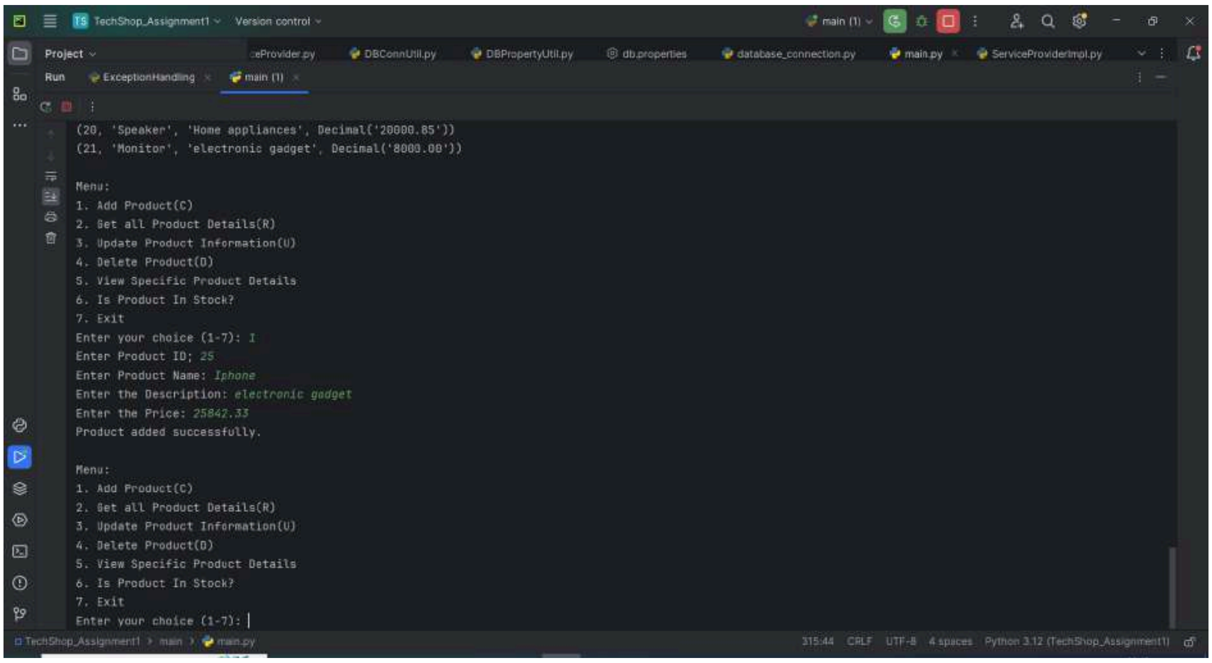
```
(8, 'Om', 'Kumar', 'om123@gmail.com', '1111111136', 'Bihar', 1)
(9, 'Roshan', 'Singh', 'roshan@gmail.com', '1111111137', 'Gandhinagar', 0)
(10, 'Arava', 'Krishnavenni', 'arava@gmail.com', '1111111139', 'Chennai', 0)
(11, 'Jasmine', 'Roy', 'jasmine123@gmail.com', '2258746984', 'Mumbai', 0)
(12, 'Ankush', 'Singh', 'ankush@gmail.com', '4587123650', 'Delhi', None)

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 4
Enter Customer ID: 12
Customer deleted successfully.

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7):
```



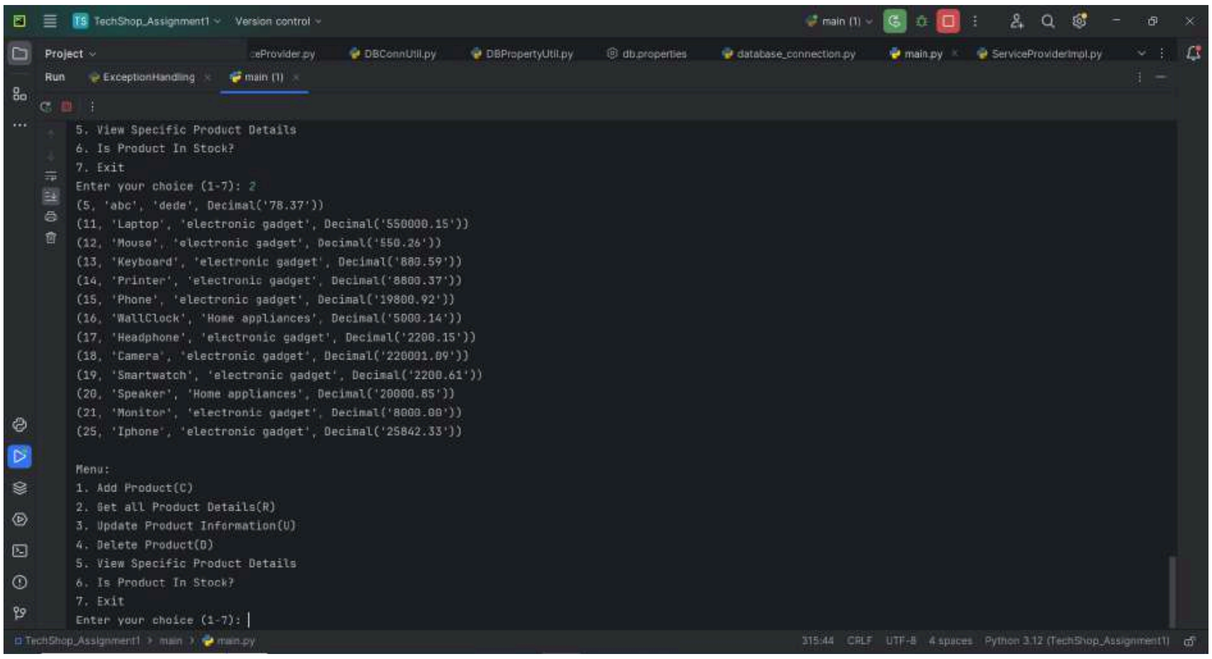
```
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 4
Enter Customer ID: 12
Customer deleted successfully.

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 6
Enter Customer ID: 5
Total orders for customer 5: 2

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the  Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7):
```



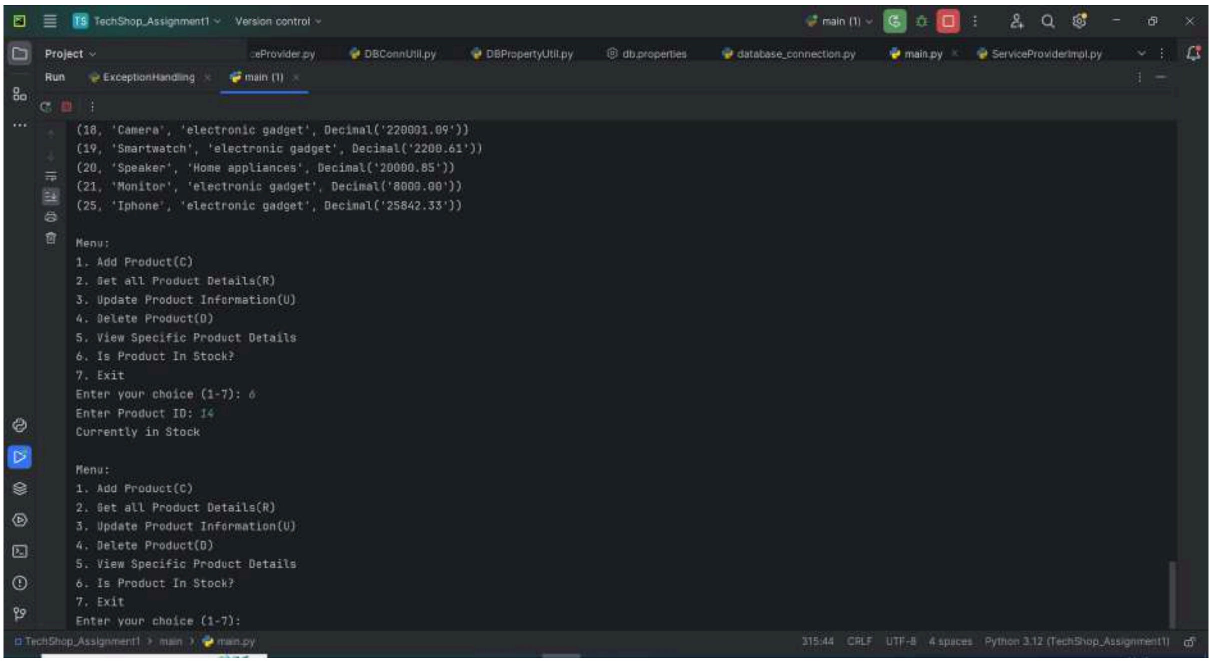
```
*****************PRODUCT MENU*****************

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 2
(5, 'abc', 'dede', Decimal('78.37'))
(11, 'Laptop', 'electronic gadget', Decimal('550000.15'))
(12, 'Mouse', 'electronic gadget', Decimal('550.26'))
(13, 'Keyboard', 'electronic gadget', Decimal('880.59'))
(14, 'Printer', 'electronic gadget', Decimal('8800.37'))
(15, 'Phone', 'electronic gadget', Decimal('19800.92'))
(16, 'WallClock', 'Home appliances', Decimal('5000.14'))
(17, 'Headphone', 'electronic gadget', Decimal('2200.15'))
(18, 'Camera', 'electronic gadget', Decimal('220001.09'))
(19, 'Smartwatch', 'electronic gadget', Decimal('2200.61'))
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))

Menu:
1. Add Product(C)
2. Get all Product Details(R)
```



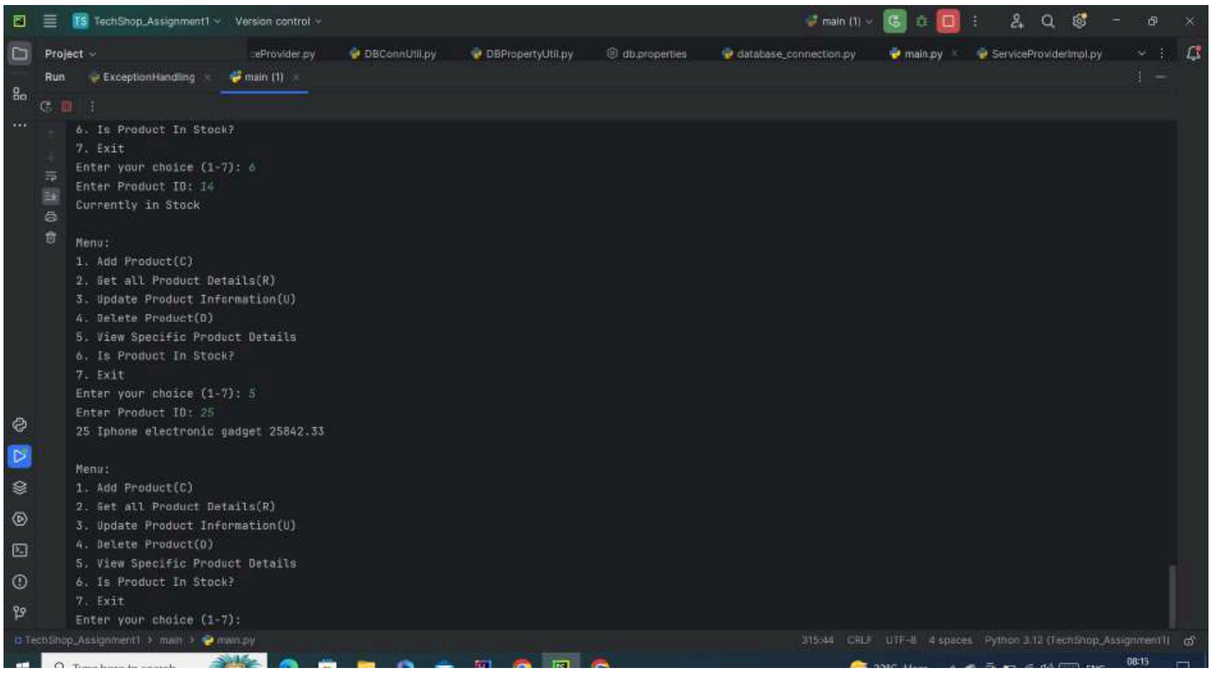
```
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 1
Enter Product ID: 25
Enter Product Name: Iphone
Enter the Description: electronic gadget
Enter the Price: 25842.33
Product added successfully.

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7):
```



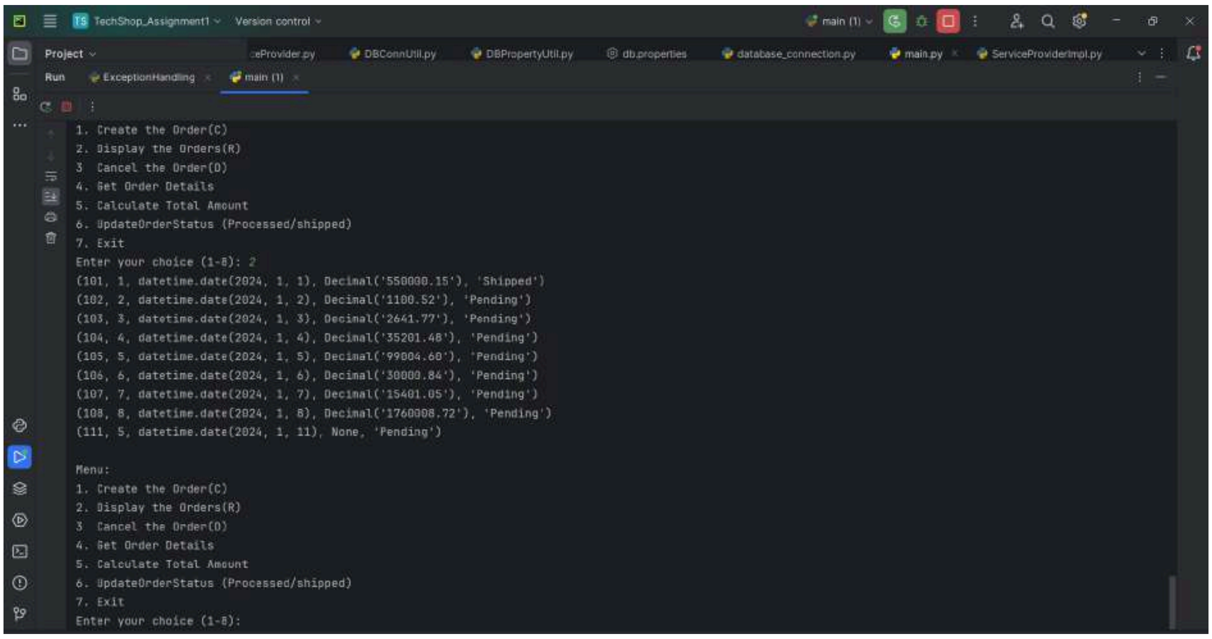
```
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 2
(5, 'abc', 'dede', Decimal('78.37'))
(11, 'Laptop', 'electronic gadget', Decimal('550000.15'))
(12, 'Mouse', 'electronic gadget', Decimal('550.26'))
(13, 'Keyboard', 'electronic gadget', Decimal('880.59'))
(14, 'Printer', 'electronic gadget', Decimal('8800.37'))
(15, 'Phone', 'electronic gadget', Decimal('19800.92'))
(16, 'WallClock', 'Home appliances', Decimal('5000.14'))
(17, 'Headphone', 'electronic gadget', Decimal('2200.15'))
(18, 'Camera', 'electronic gadget', Decimal('220001.09'))
(19, 'Smartwatch', 'electronic gadget', Decimal('2200.61'))
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))
(25, 'Iphone', 'electronic gadget', Decimal('25842.33'))

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7):
```



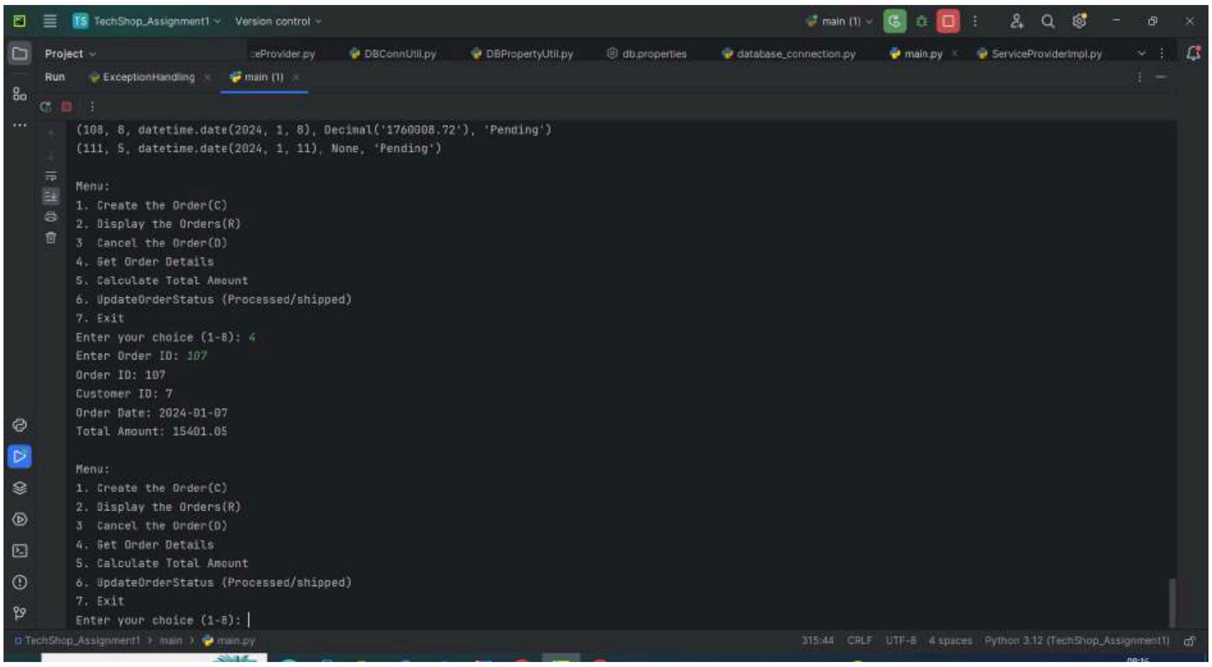
```
(18, 'Camera', 'electronic gadget', Decimal('220001.09'))
(19, 'Smartwatch', 'electronic gadget', Decimal('2200.61'))
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))
(25, 'Iphone', 'electronic gadget', Decimal('25842.33'))

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 6
Enter Product ID: 14
Currently in Stock

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7):
```



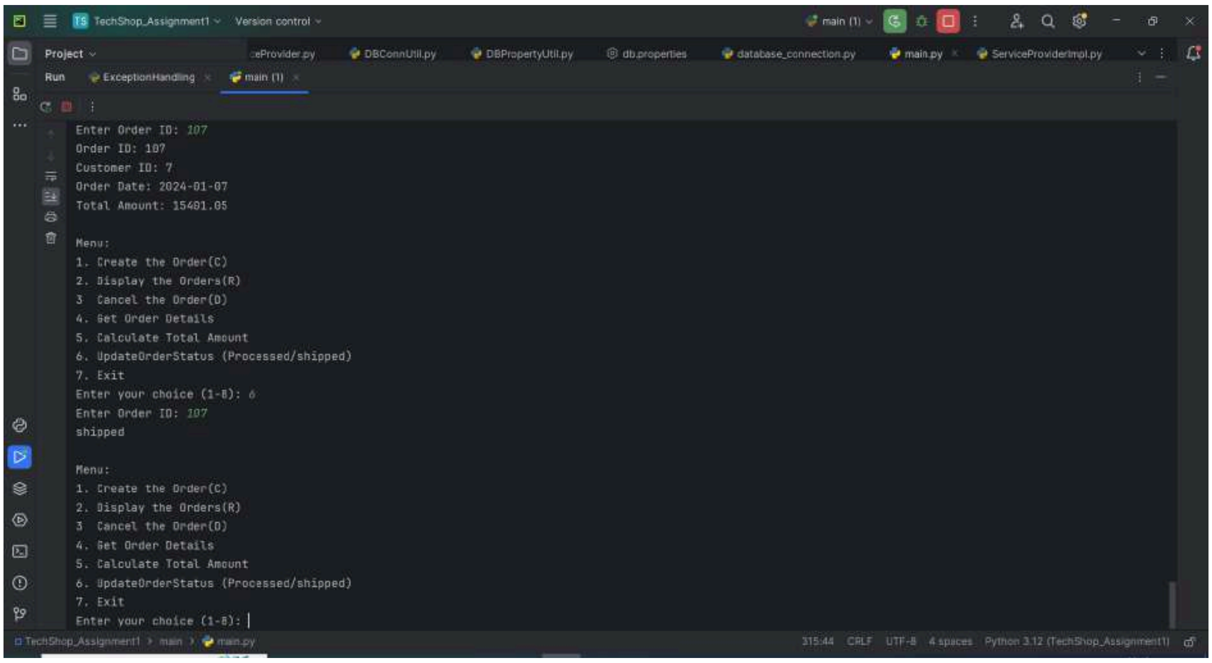
```
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 6
Enter Product ID: 14
Currently in Stock

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 5
Enter Product ID: 25
25 Iphone electronic gadget 25842.33

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7):
```



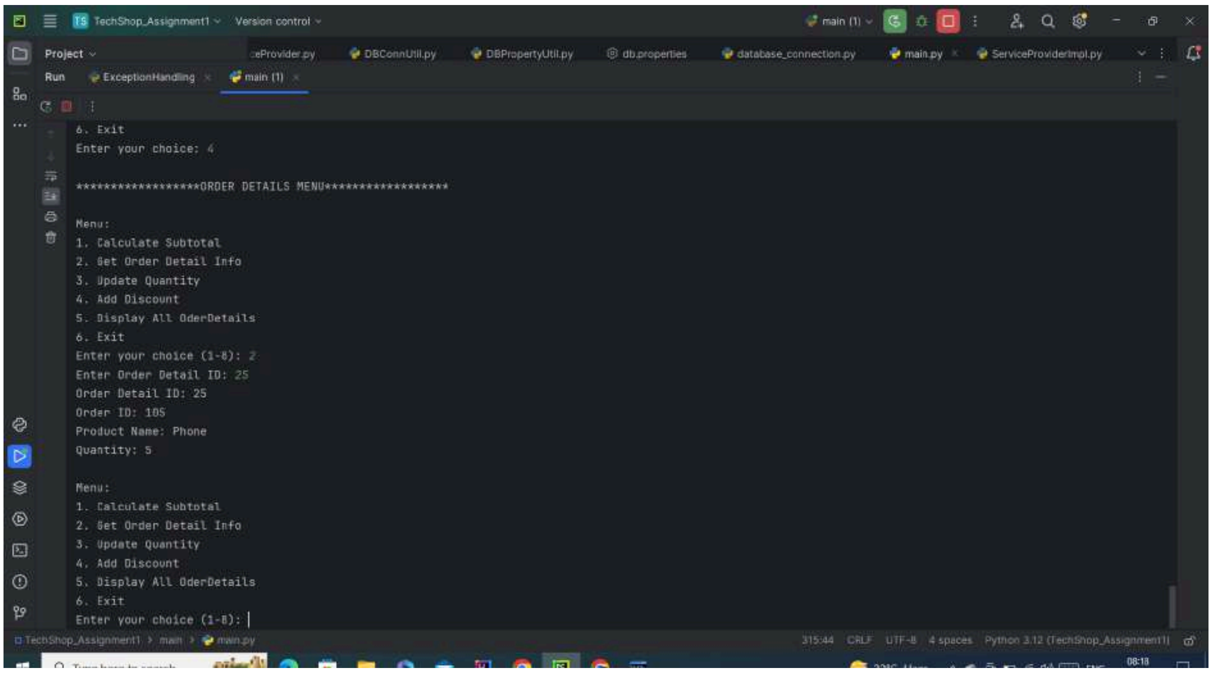
```
1. Create the Order(C)
2. Display the Orders(R)
3  Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): 2
(101, 1, datetime.date(2024, 1, 1), Decimal('550000.15'), 'Shipped')
(102, 2, datetime.date(2024, 1, 2), Decimal('1100.52'), 'Pending')
(103, 3, datetime.date(2024, 1, 3), Decimal('2641.77'), 'Pending')
(104, 4, datetime.date(2024, 1, 4), Decimal('35201.48'), 'Pending')
(105, 5, datetime.date(2024, 1, 5), Decimal('99004.60'), 'Pending')
(106, 6, datetime.date(2024, 1, 6), Decimal('30000.84'), 'Pending')
(107, 7, datetime.date(2024, 1, 7), Decimal('15401.05'), 'Pending')
(108, 8, datetime.date(2024, 1, 8), Decimal('1760008.72'), 'Pending')
(111, 5, datetime.date(2024, 1, 11), None, 'Pending')

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3  Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8):
```

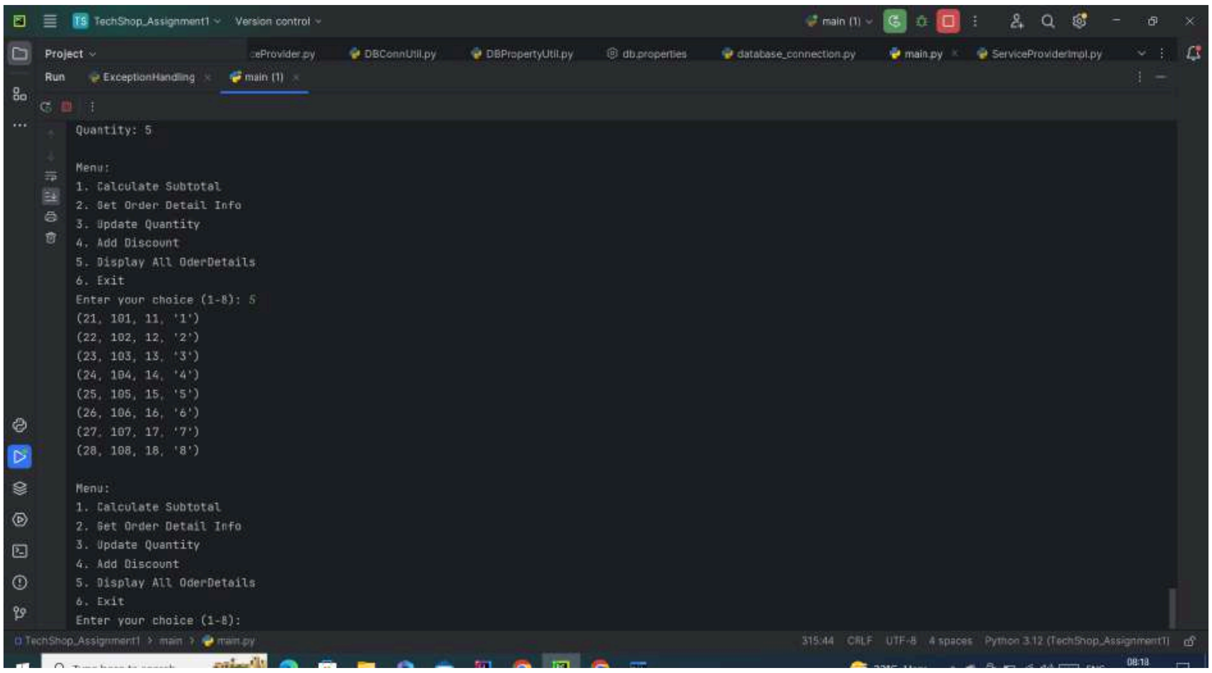


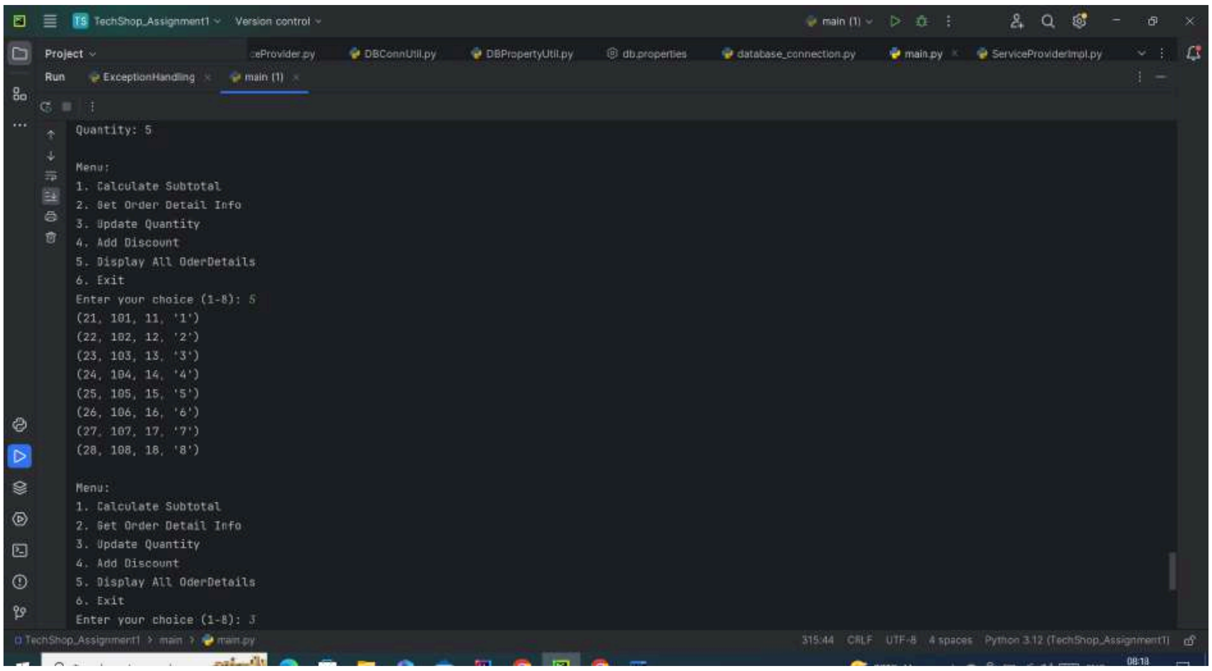
```
(108, 8, datetime.date(2024, 1, 8), Decimal('1760808.72'), 'Pending')
(111, 5, datetime.date(2024, 1, 11), None, 'Pending')

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3  Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): 4
Enter Order ID: 107
Order ID: 107
Customer ID: 7
Order Date: 2024-01-07
Total Amount: 15401.05

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3  Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8):
```



```
Enter Order ID: 107
Order ID: 107
Customer ID: 7
Order Date: 2024-01-07
Total Amount: 15401.05

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3  Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): 6
Enter Order ID: 107
shipped

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3  Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8):
```


```
6. Exit
Enter your choice: 4

******************ORDER DETAILS MENU******************

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8): 2
Enter Order Detail ID: 25
Order Detail ID: 25
Order ID: 105
Product Name: Phone
Quantity: 5

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8):
```


```
Quantity: 5

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8): 5
(21, 101, 11, '1')
(22, 102, 12, '2')
(23, 103, 13, '3')
(24, 104, 14, '4')
(25, 105, 15, '5')
(26, 106, 16, '6')
(27, 107, 17, '7')
(28, 108, 18, '8')

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8):
```



```
Quantity: 5

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8): 5
(21, 101, 11, '1')
(22, 102, 12, '2')
(23, 103, 13, '3')
(24, 104, 14, '4')
(25, 105, 15, '5')
(26, 106, 16, '6')
(27, 107, 17, '7')
(28, 108, 18, '8')

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OderDetails
6. Exit
Enter your choice (1-8): 3
```