**ASSIGNMENT 1 PYTHON**

**NAME :- SUSHANT KUMAR SINGH**

**Project:- (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:

**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:

Attributes:

• CustomerID (int)

• FirstName (string)

• LastName (string)

• Email (string)

• Phone (string)

• Address (string)

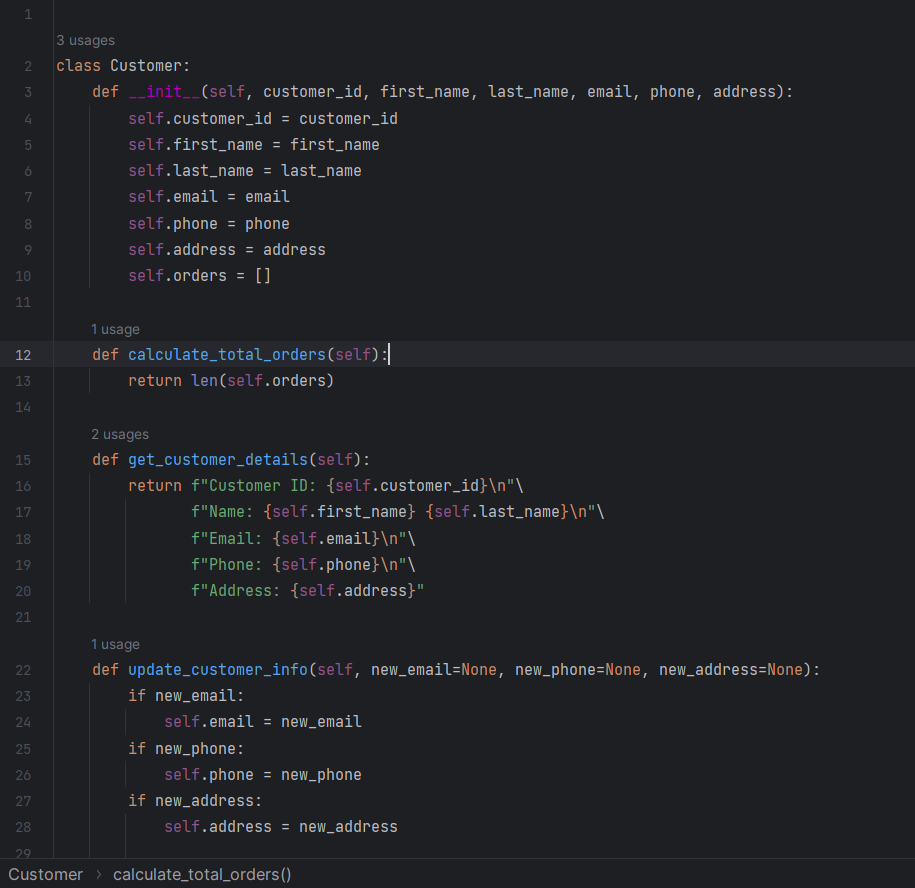
Methods:

• CalculateTotalOrders(): Calculates the total number of orders placed by this customer.

• GetCustomerDetails(): Retrieves and displays detailed information about the customer.

• UpdateCustomerInfo(): Allows the customer to update their information (e.g., email, phone, or

address).



**Products Class:**

Attributes:

• ProductID (int)

• ProductName (string)

• Description (string)

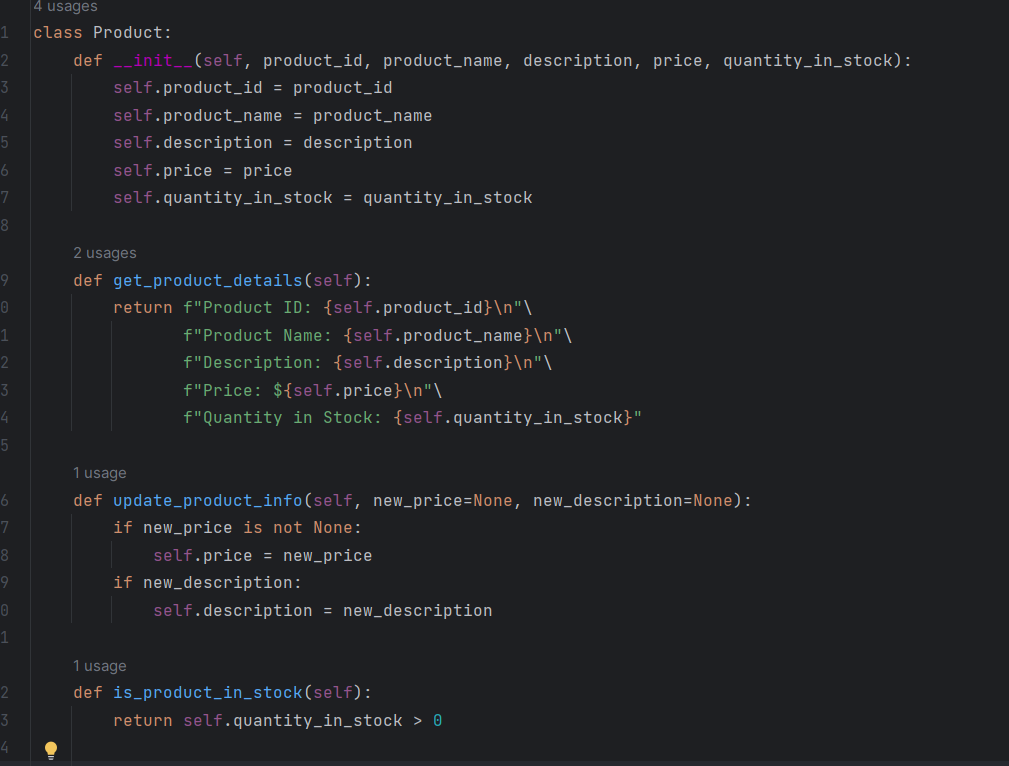
• Price (decimal)

Methods:

• GetProductDetails(): Retrieves and displays detailed information about the product.

• UpdateProductInfo(): Allows updates to product details (e.g., price, description).

• IsProductInStock(): Checks if the product is currently in stock.



**Orders Class:**

Attributes:

• OrderID (int)

• Customer (Customer) - Use composition to reference the Customer who placed the order.

• OrderDate (DateTime)

• TotalAmount (decimal)

Methods:

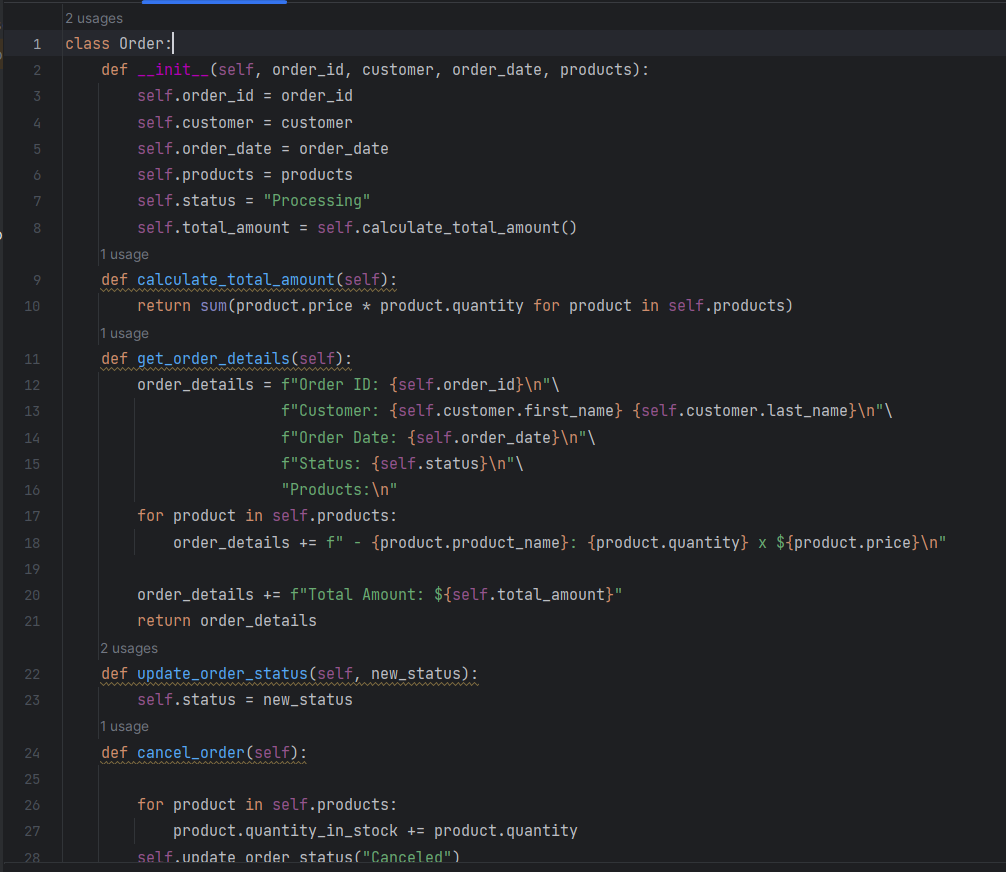
• CalculateTotalAmount() - Calculate the total amount of the order.

• GetOrderDetails(): Retrieves and displays the details of the order (e.g., product list and

quantities).

• UpdateOrderStatus(): Allows updating the status of the order (e.g., processing, shipped).

• CancelOrder(): Cancels the order and adjusts stock levels for products.



**OrderDetails Class:**

Attributes:

• OrderDetailID (int)

• Order (Order) - Use composition to reference the Order to which this detail belongs.

• Product (Product) - Use composition to reference the Product included in the order detail.

• Quantity (int)

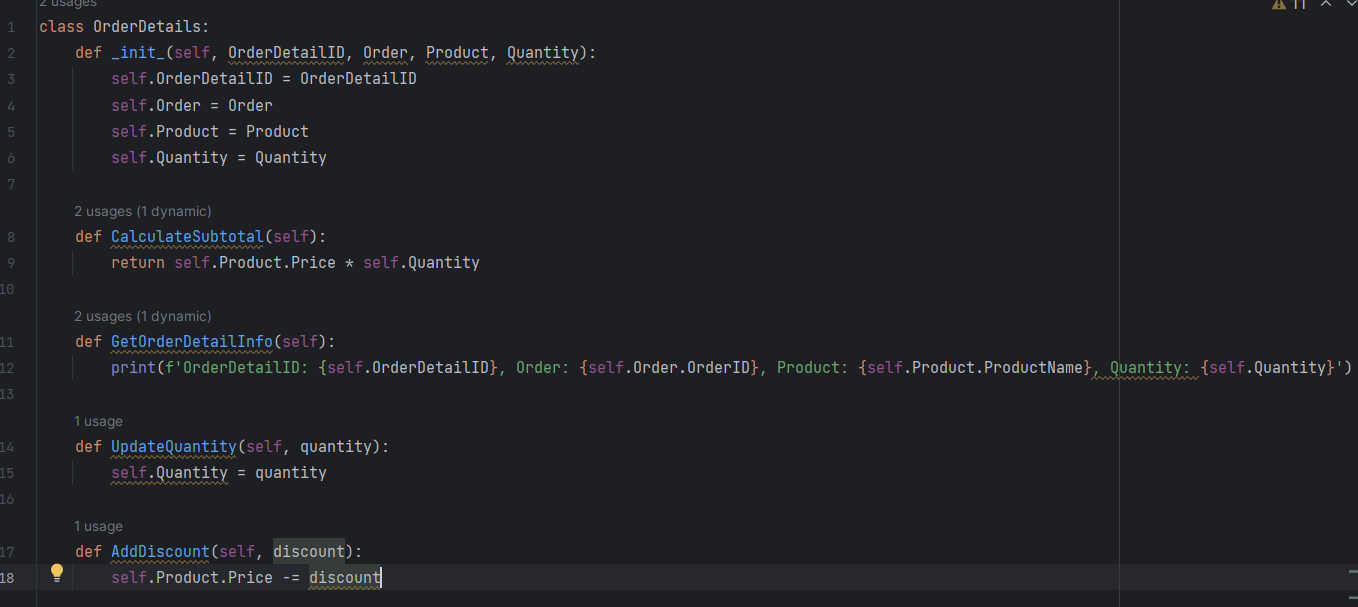
Methods:

• CalculateSubtotal() - Calculate the subtotal for this order detail.

• GetOrderDetailInfo(): Retrieves and displays information about this order detail.

• UpdateQuantity(): Allows updating the quantity of the product in this order detail.

• AddDiscount(): Applies a discount to this order detail.



**Inventory class:**

Attributes:

• InventoryID(int)

• Product (Composition): The product associated with the inventory item.

• QuantityInStock: The quantity of the product currently in stock.

• LastStockUpdate

Methods:

• GetProduct(): A method to retrieve the product associated with this inventory item.

• GetQuantityInStock(): A method to get the current quantity of the product in stock.

• AddToInventory(int quantity): A method to add a specified quantity of the product to the

inventory.

• RemoveFromInventory(int quantity): A method to remove a specified quantity of the product

from the inventory.

• UpdateStockQuantity(int newQuantity): A method to update the stock quantity to a new value.

• IsProductAvailable(int quantityToCheck): A method to check if a specified quantity of the

product is available in the inventory.

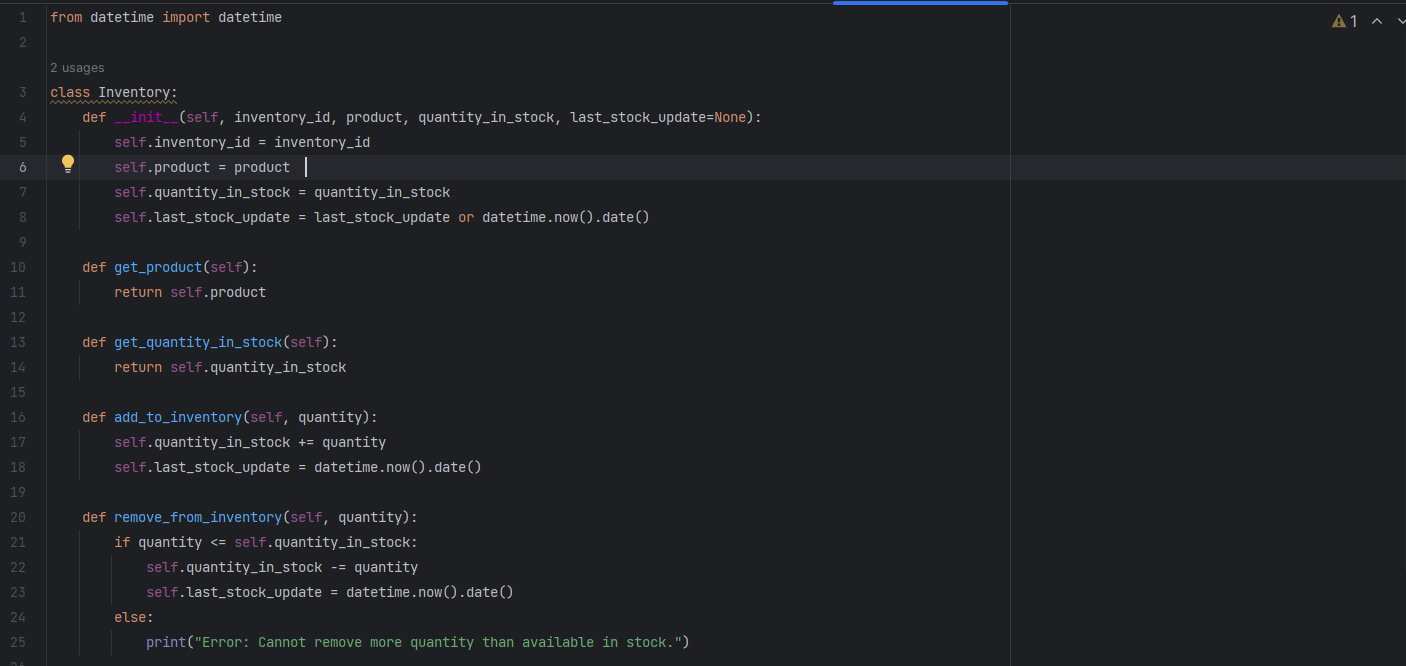
• GetInventoryValue(): A method to calculate the total value of the products in the inventory

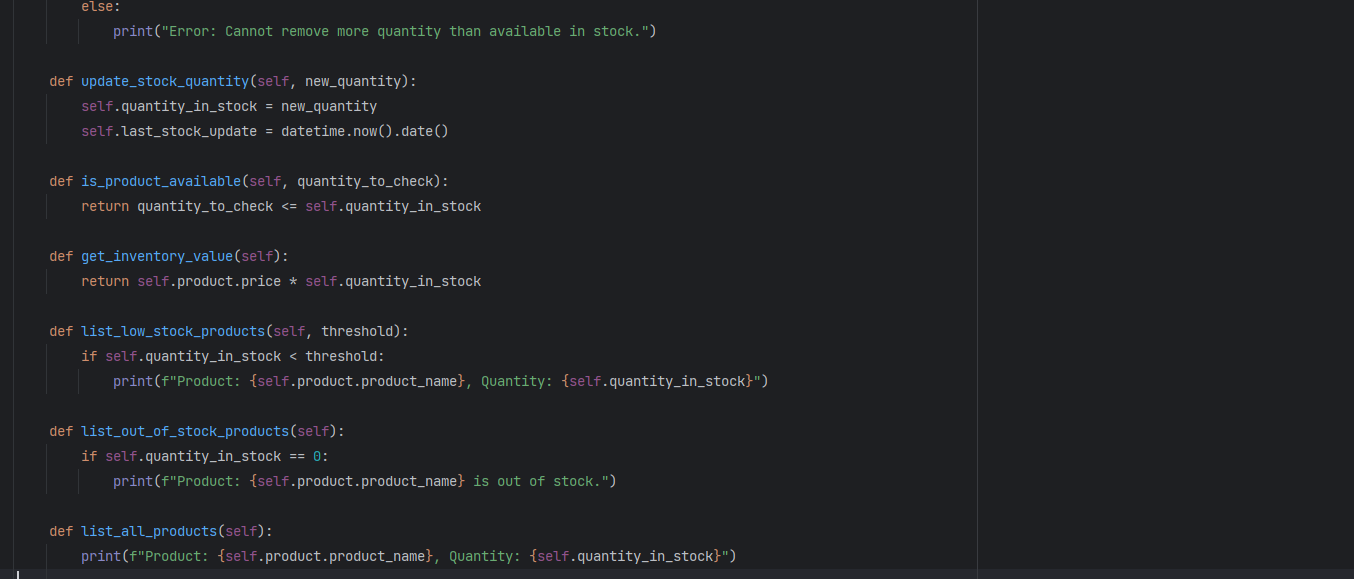
based on their prices and quantities.

• ListLowStockProducts(int threshold): A method to list products with quantities below a specified

threshold, indicating low stock.

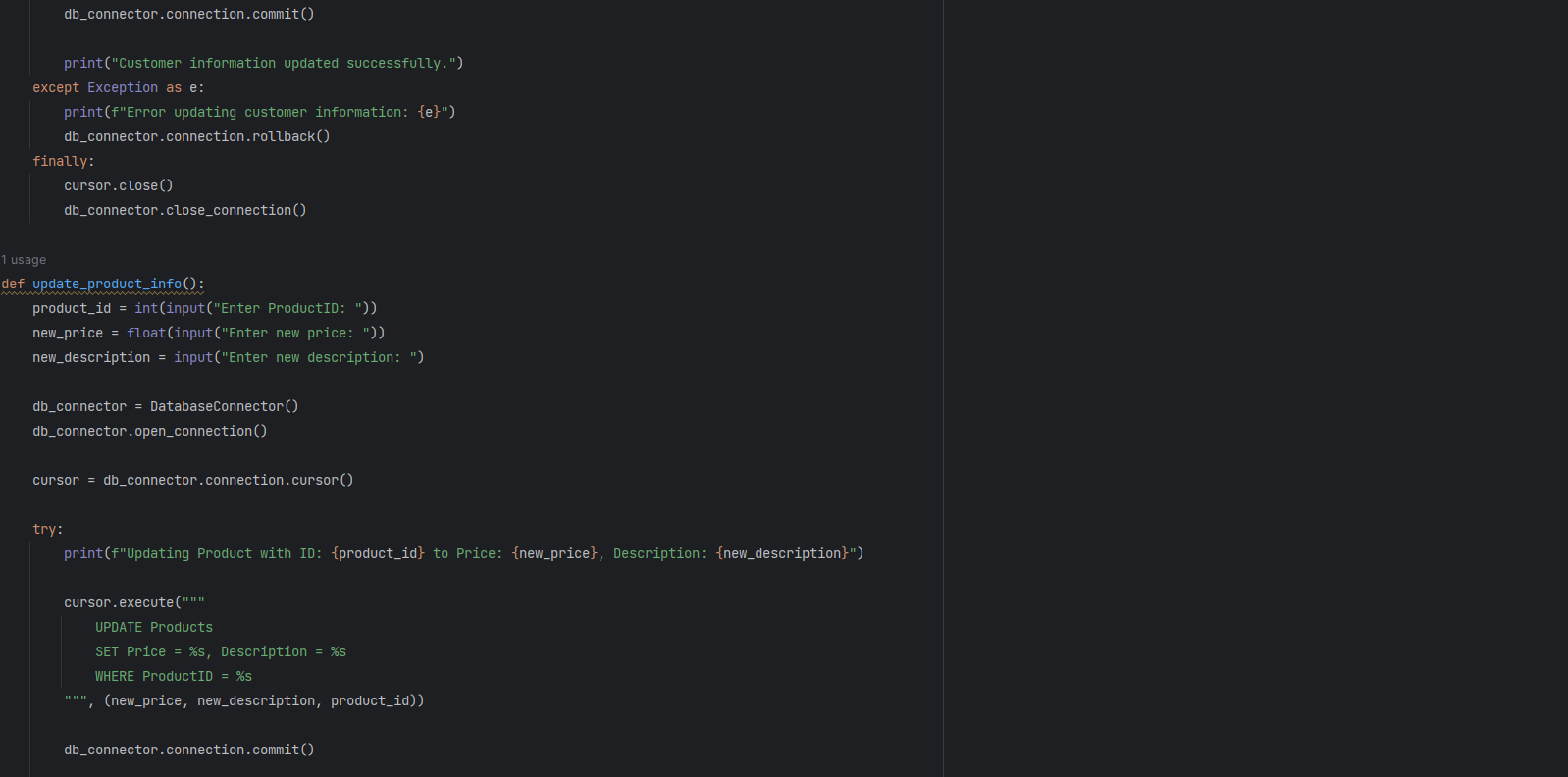
• ListOutOfStockProducts(): A method to list products that are out of stock.

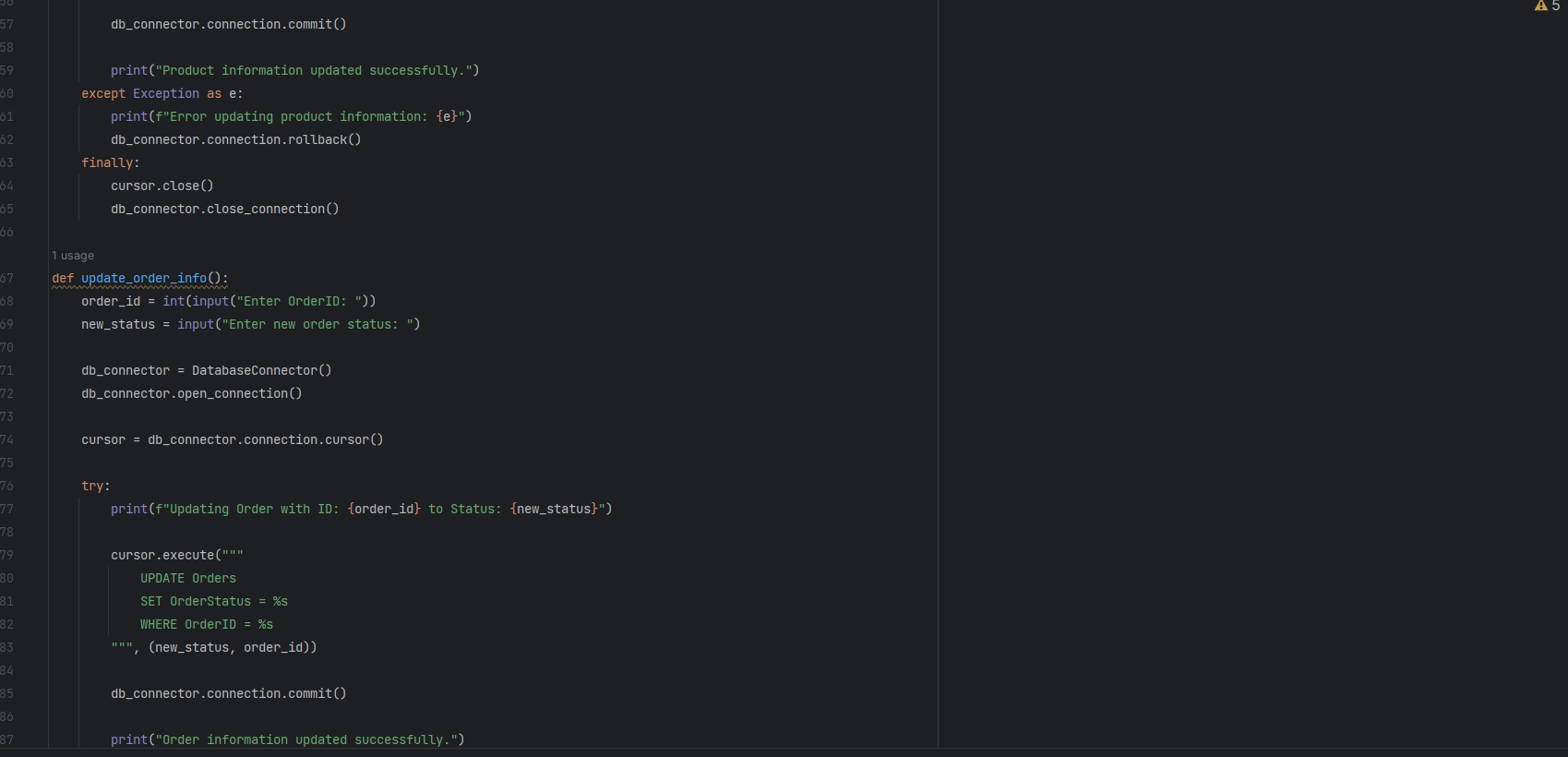


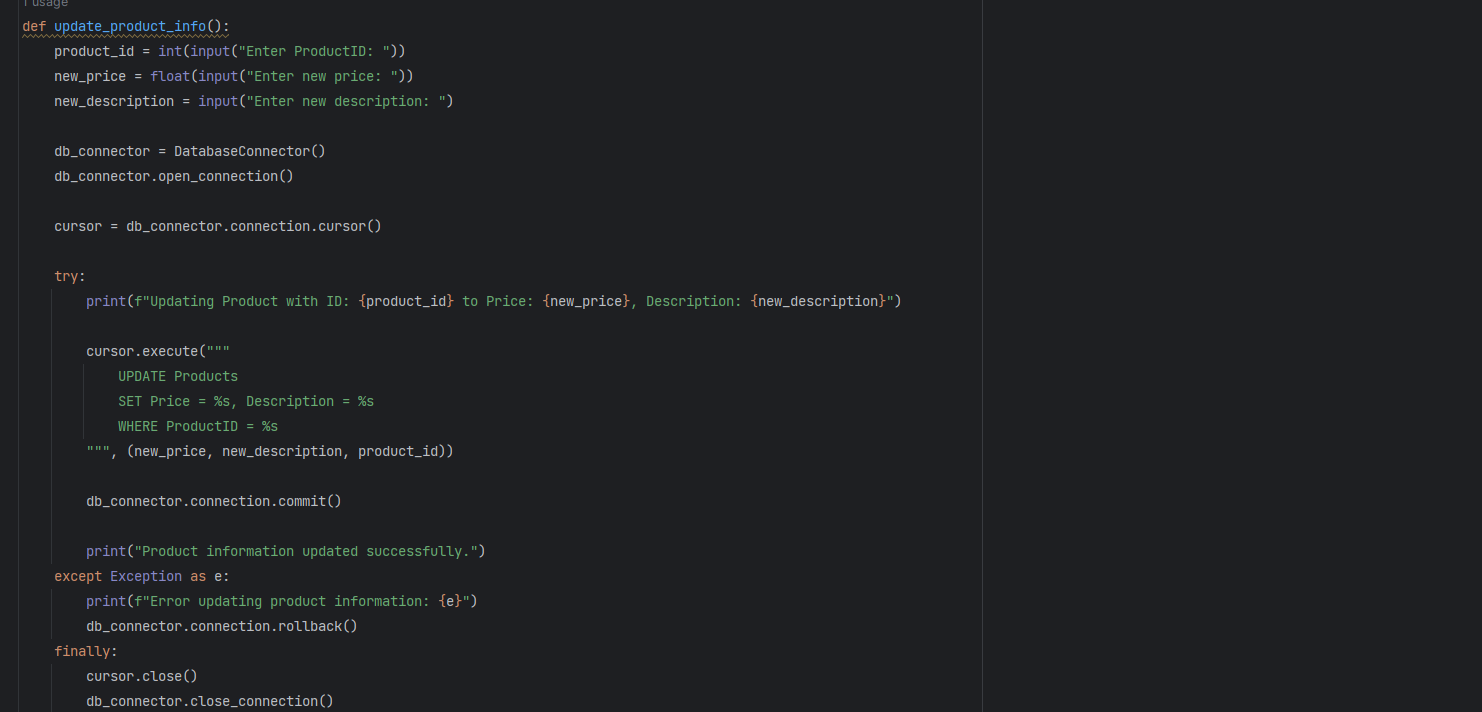


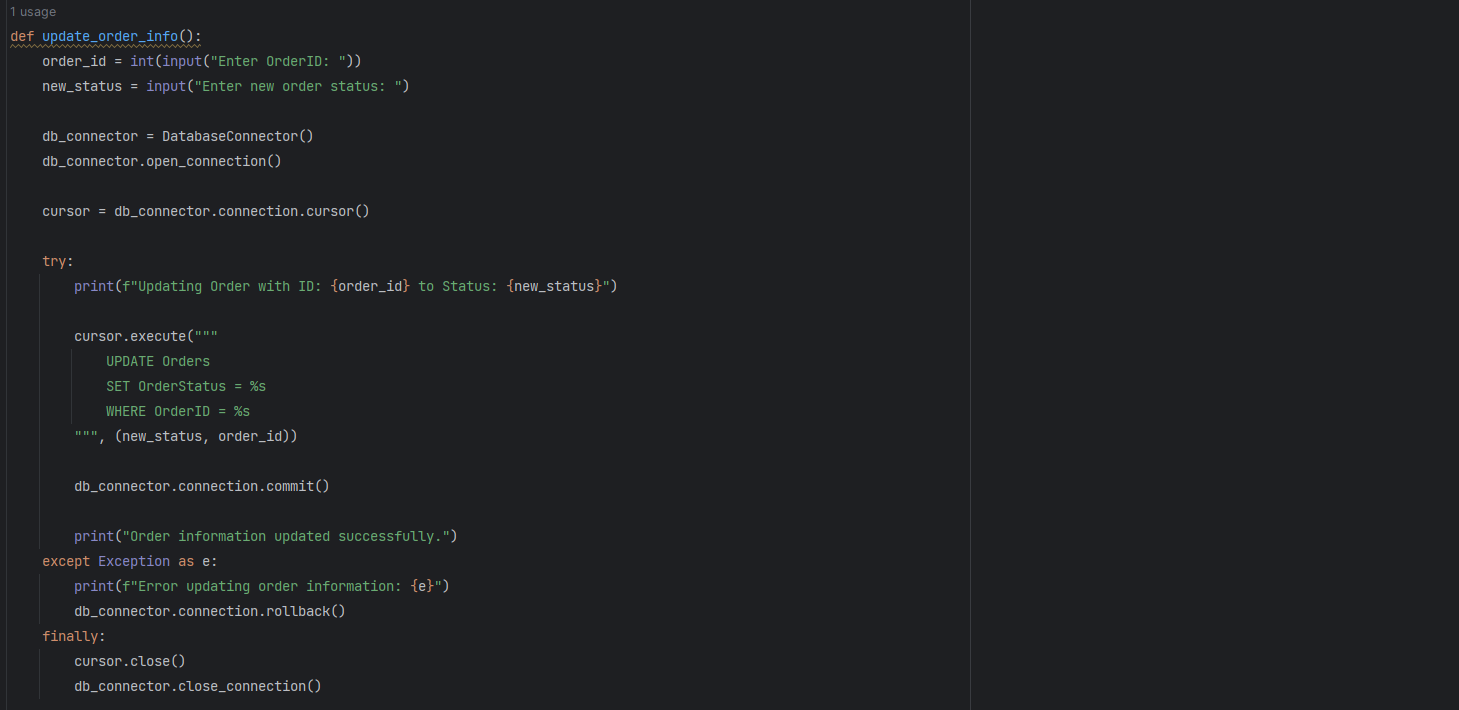
**Main.py class**

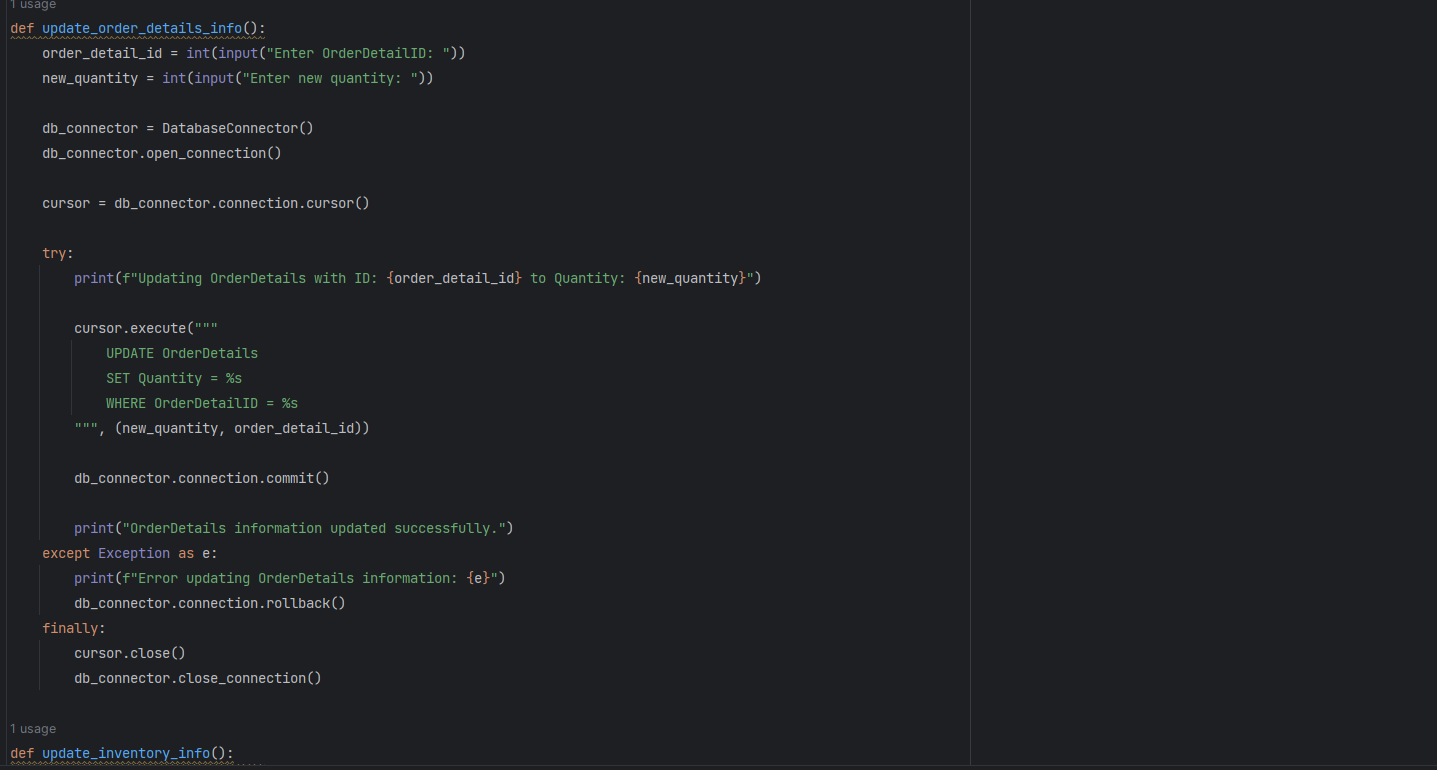
****

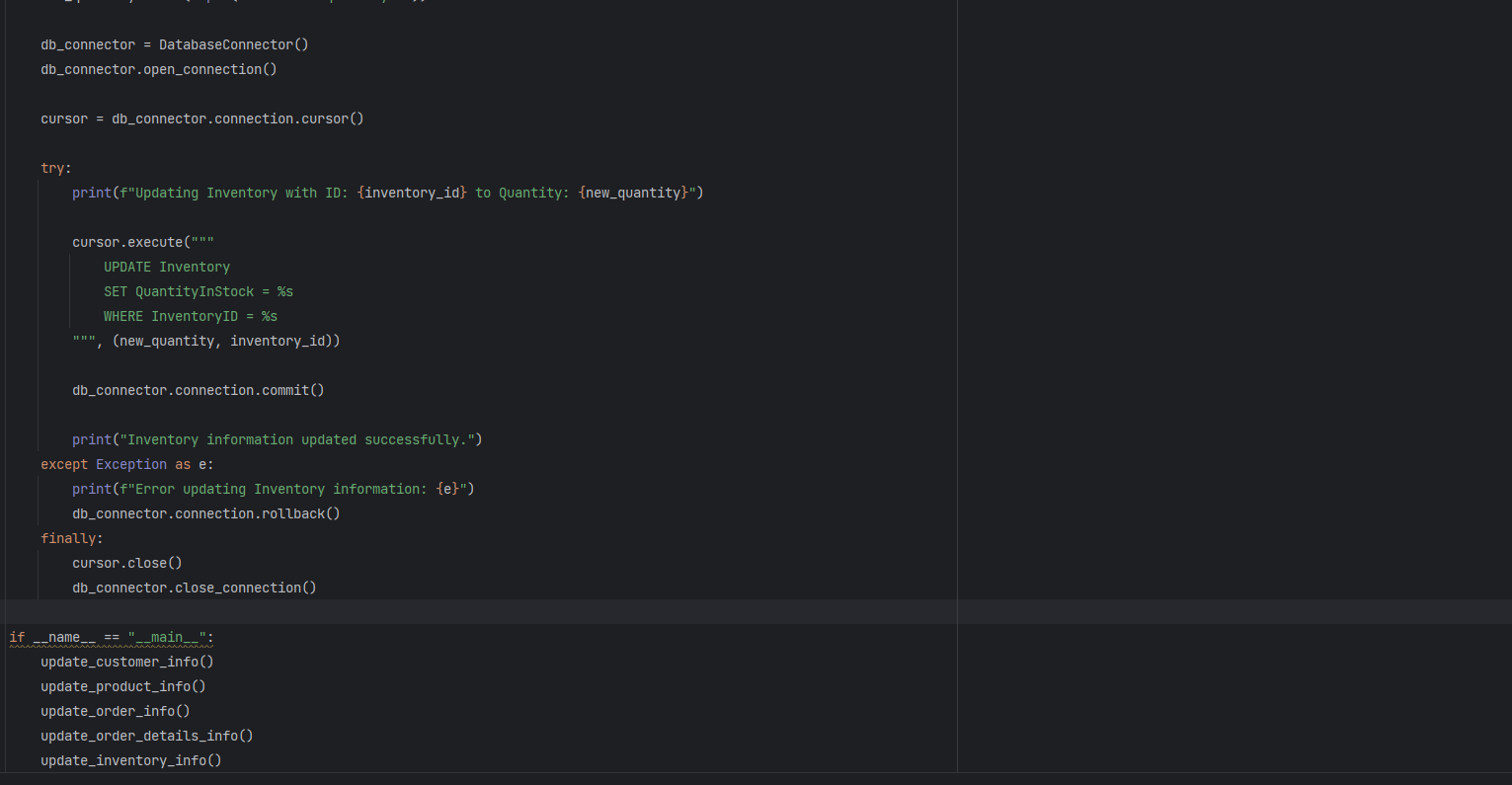
****

****

****

****

****

****

**Task 3: Encapsulation:**

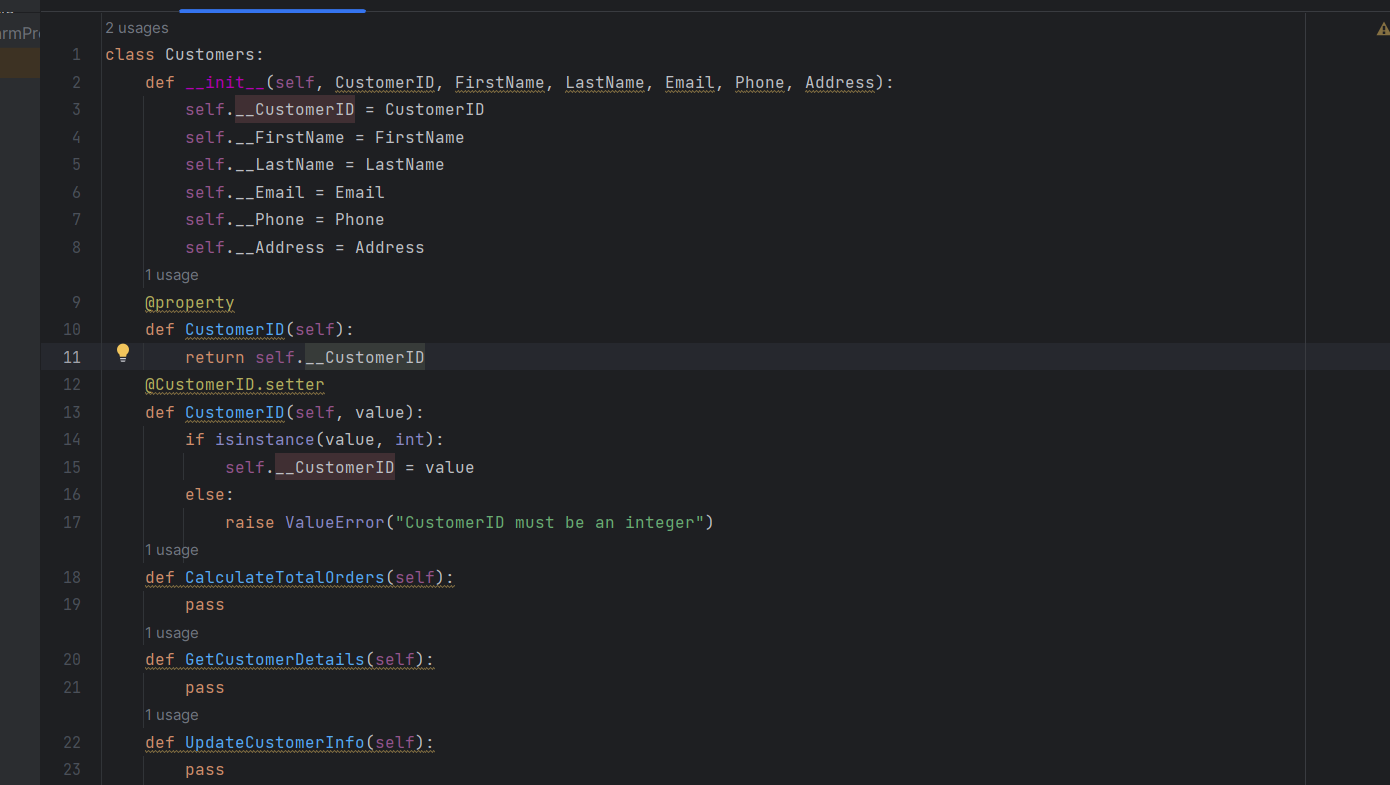
• Implement encapsulation by making the attributes private and providing public properties

(getters and setters) for each attribute.

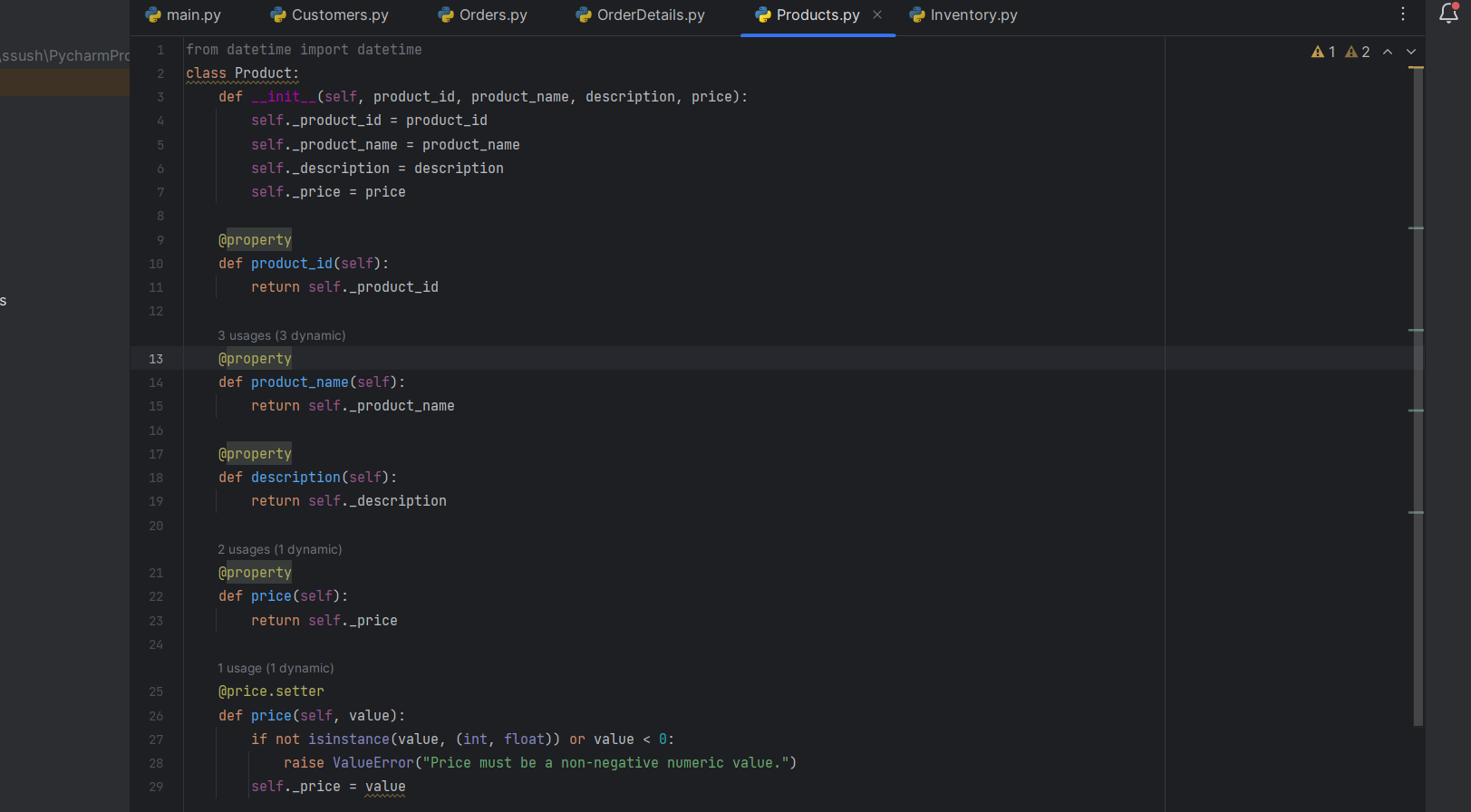
• Add data validation logic to setter methods (e.g., ensure that prices are non-negative, quantities

are positive integers).

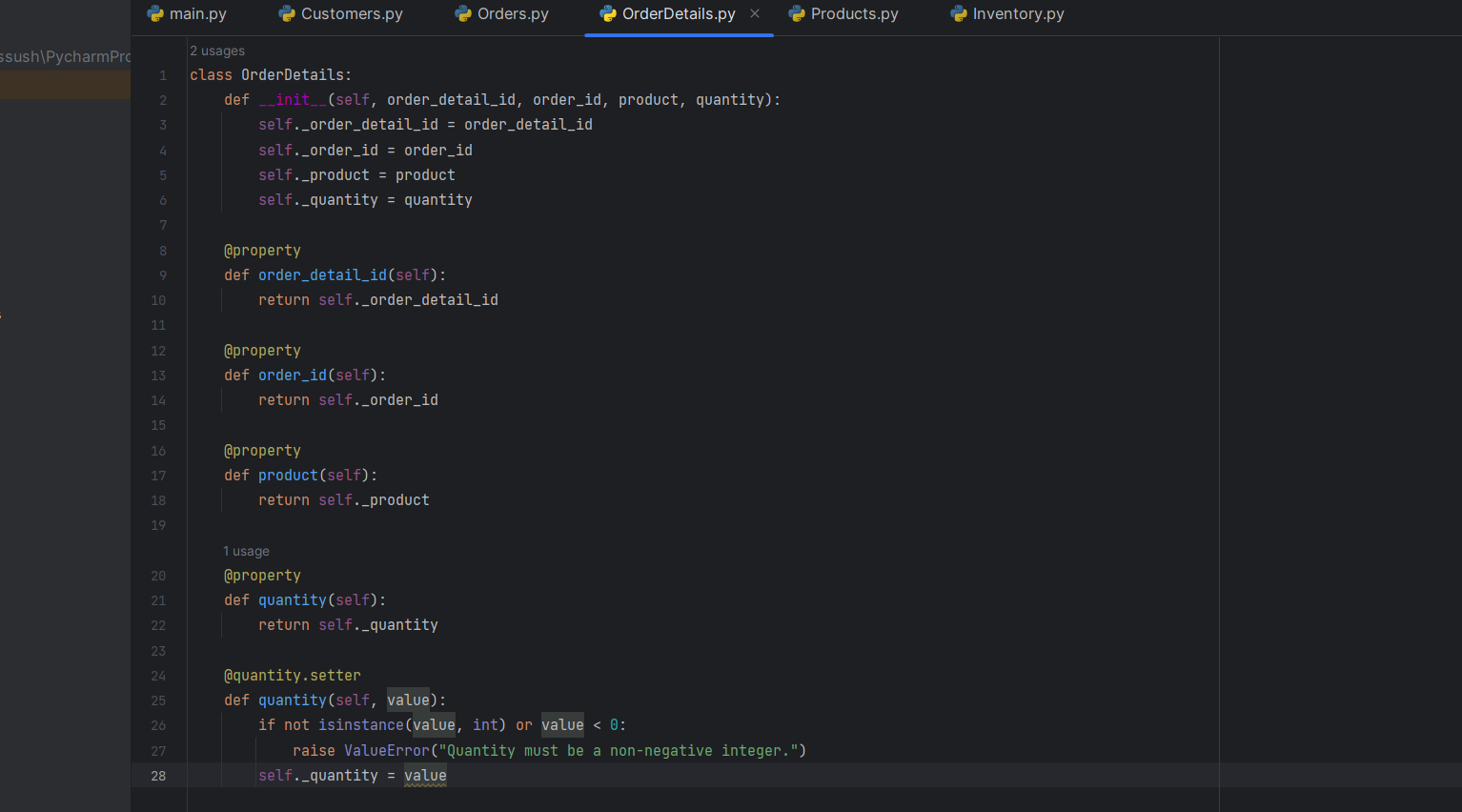
**Customer .py class with Encapsulation**



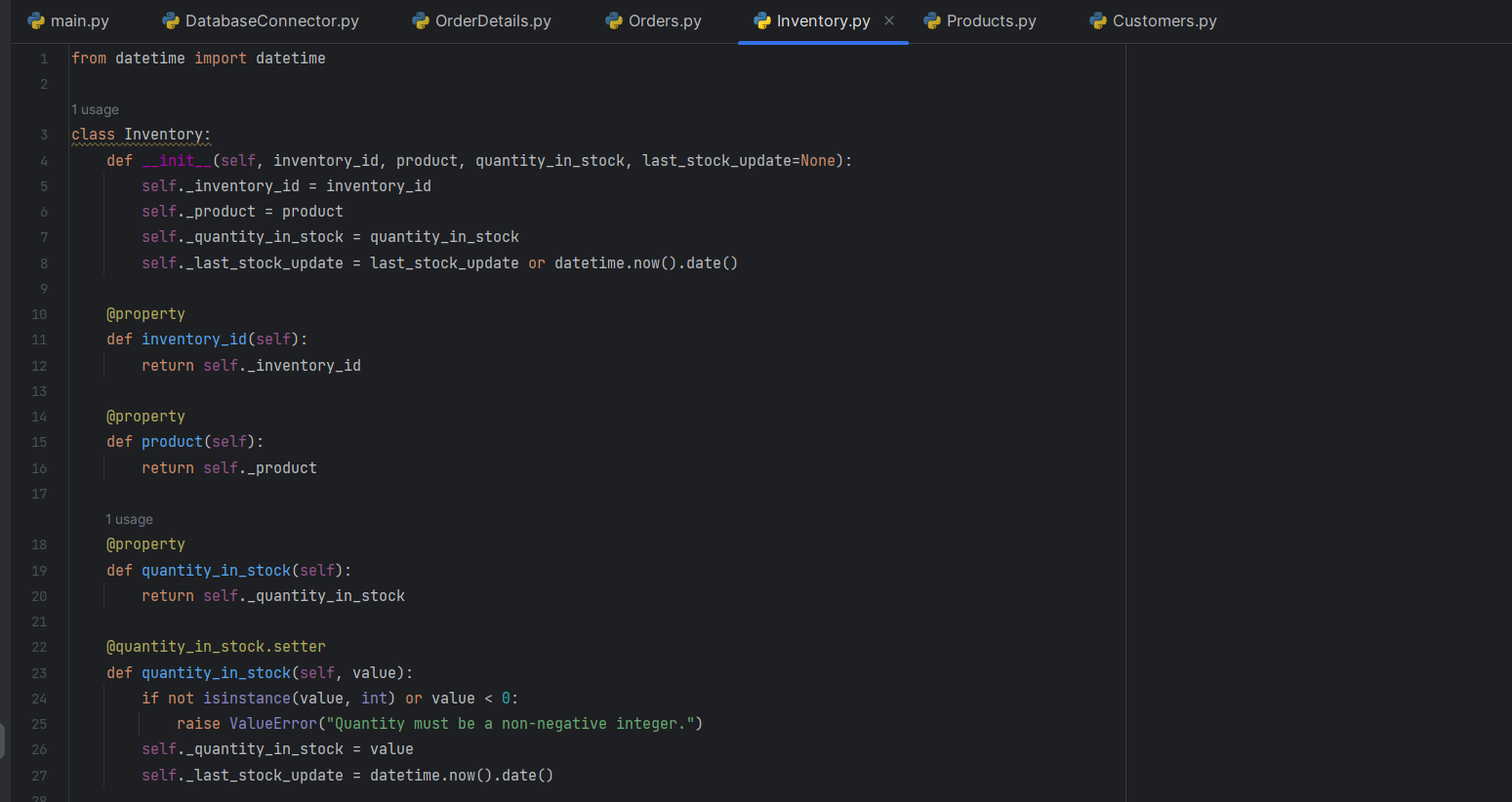
**Products.py class with Encapsulation Properties**



OrderDetails.py class with Encapsulation



**Inventory.py class with Encapsulation Properties**



**Task 4: Composition:**

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

Product objects.

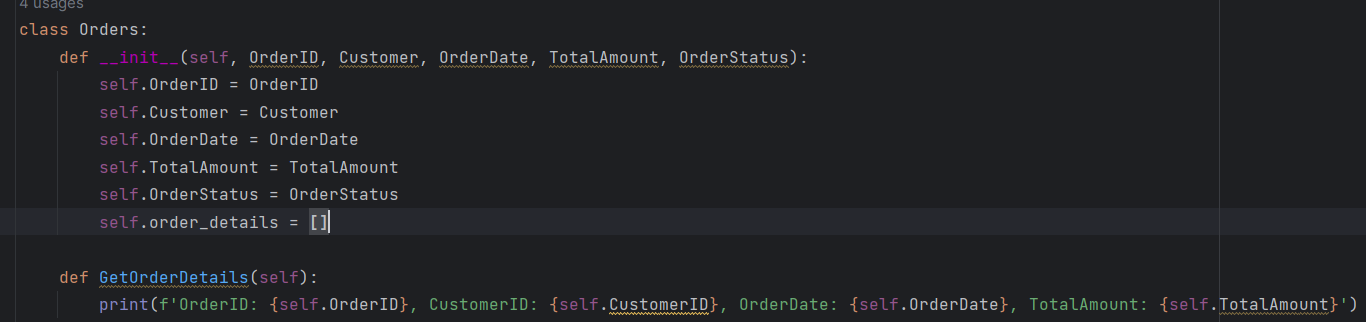
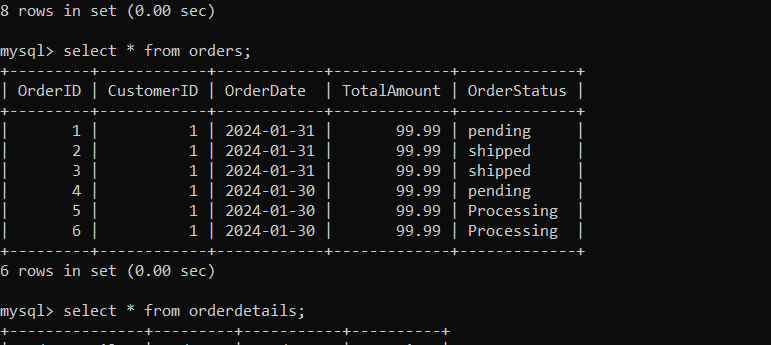
• Orders Class with Composition:

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

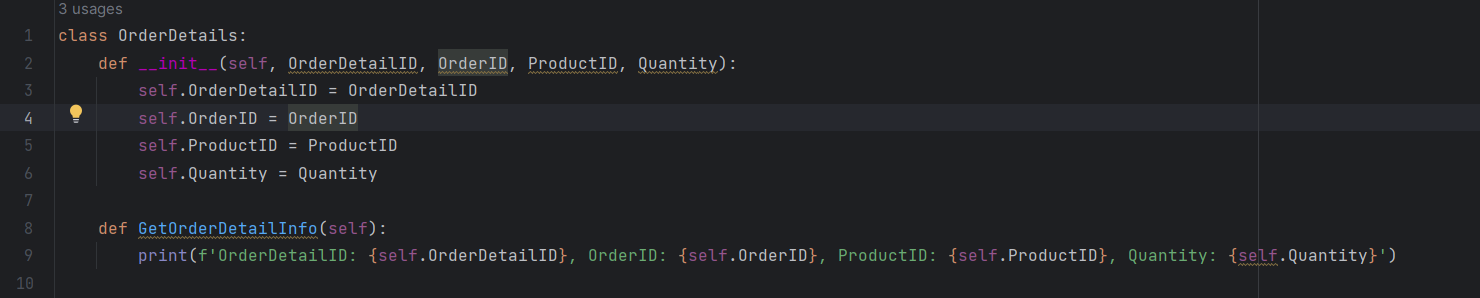
Customers object associated with the order.

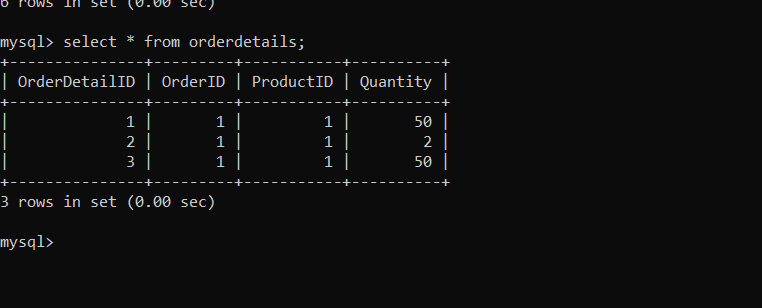
• **OrderDetails Class with Composition:**

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. TheOrder 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.



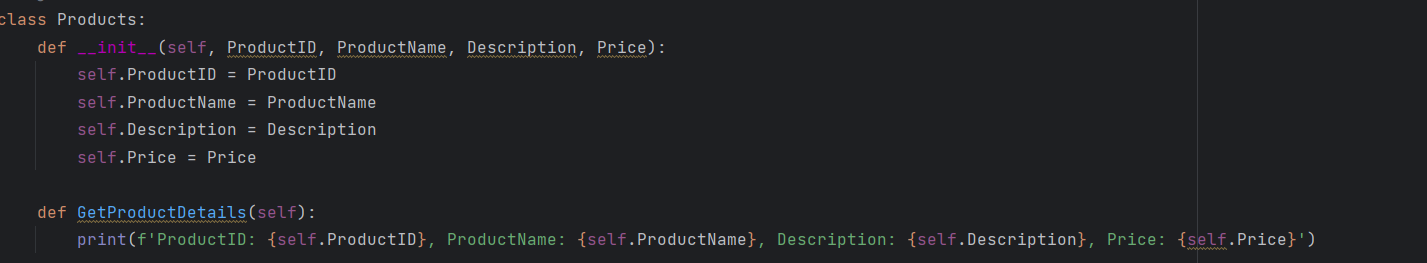


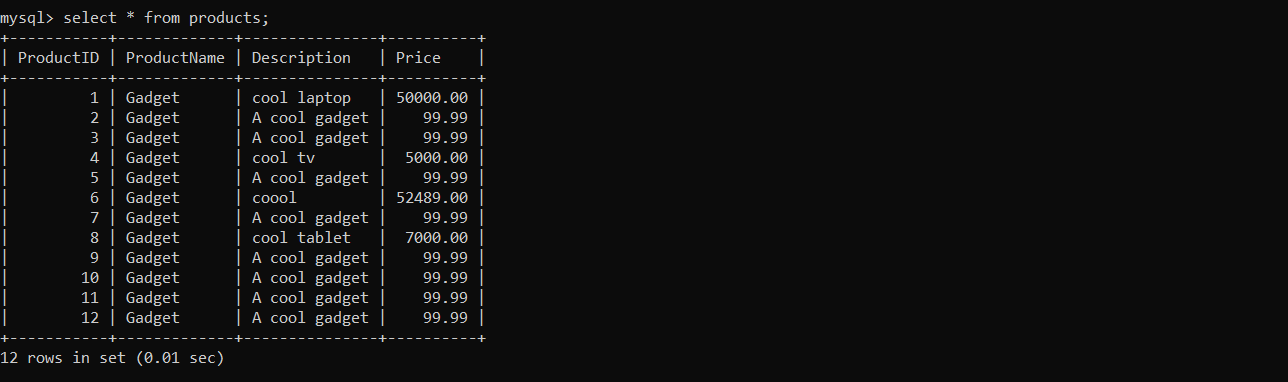
**• Customers and Products Classes:**

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.



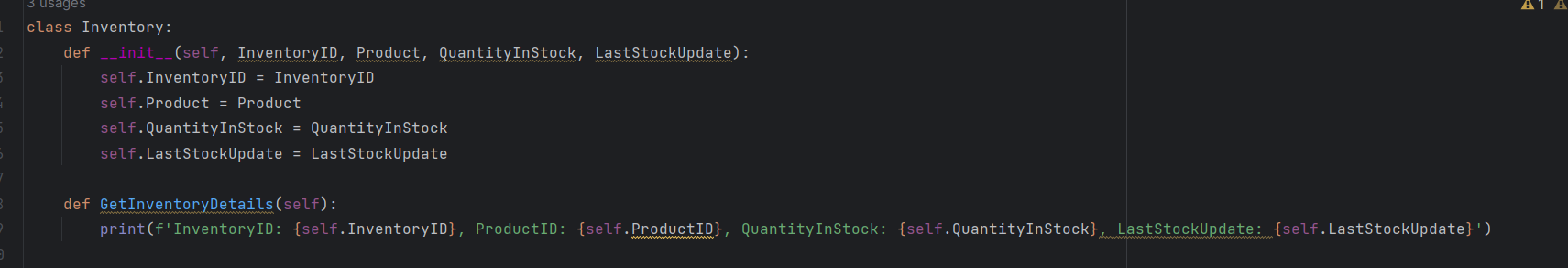


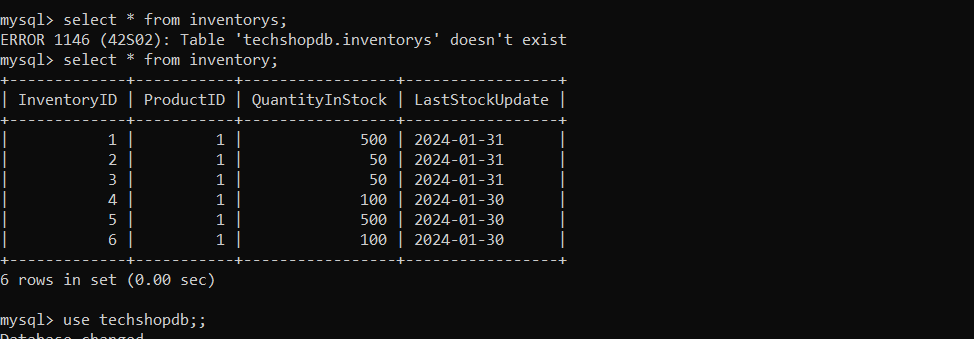
• **Inventory Class:**

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.





**Task 5: Exceptions handling**

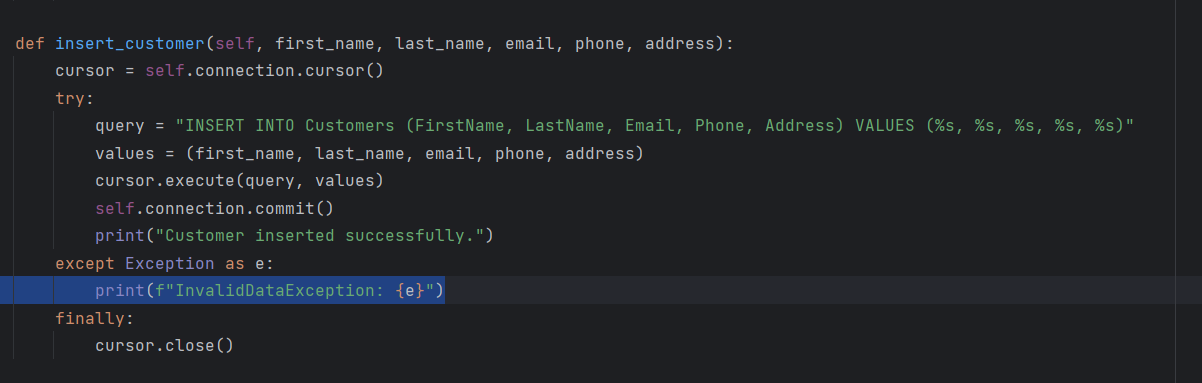
**• Data Validation:**

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

order placement).

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

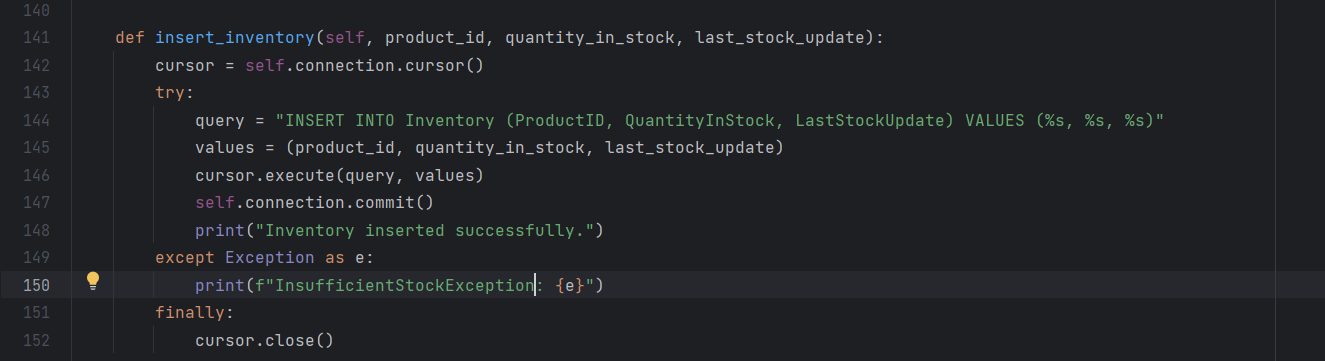
o Exception Handling: Throw a custom InvalidDataException with a clear error message.



**• Inventory Management:**

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

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

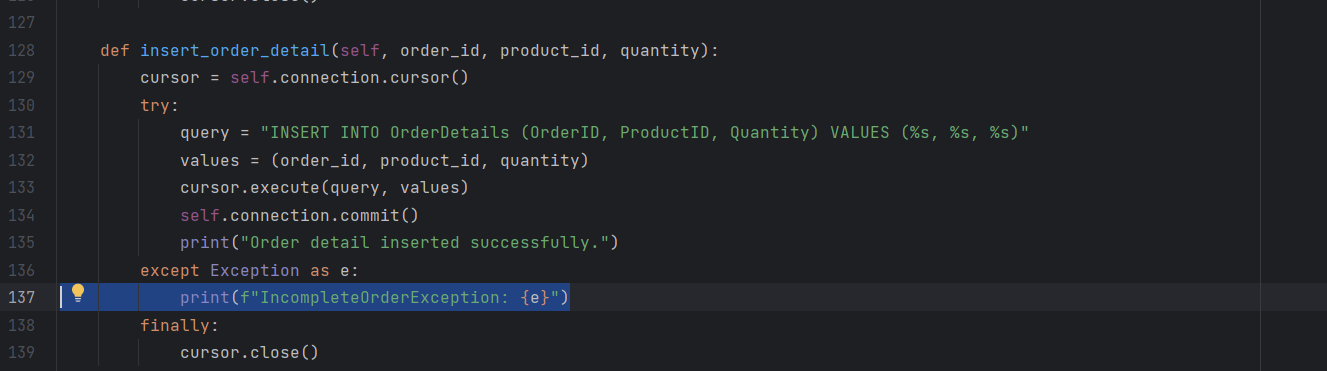


**• Order Processing:**

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

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

issue.

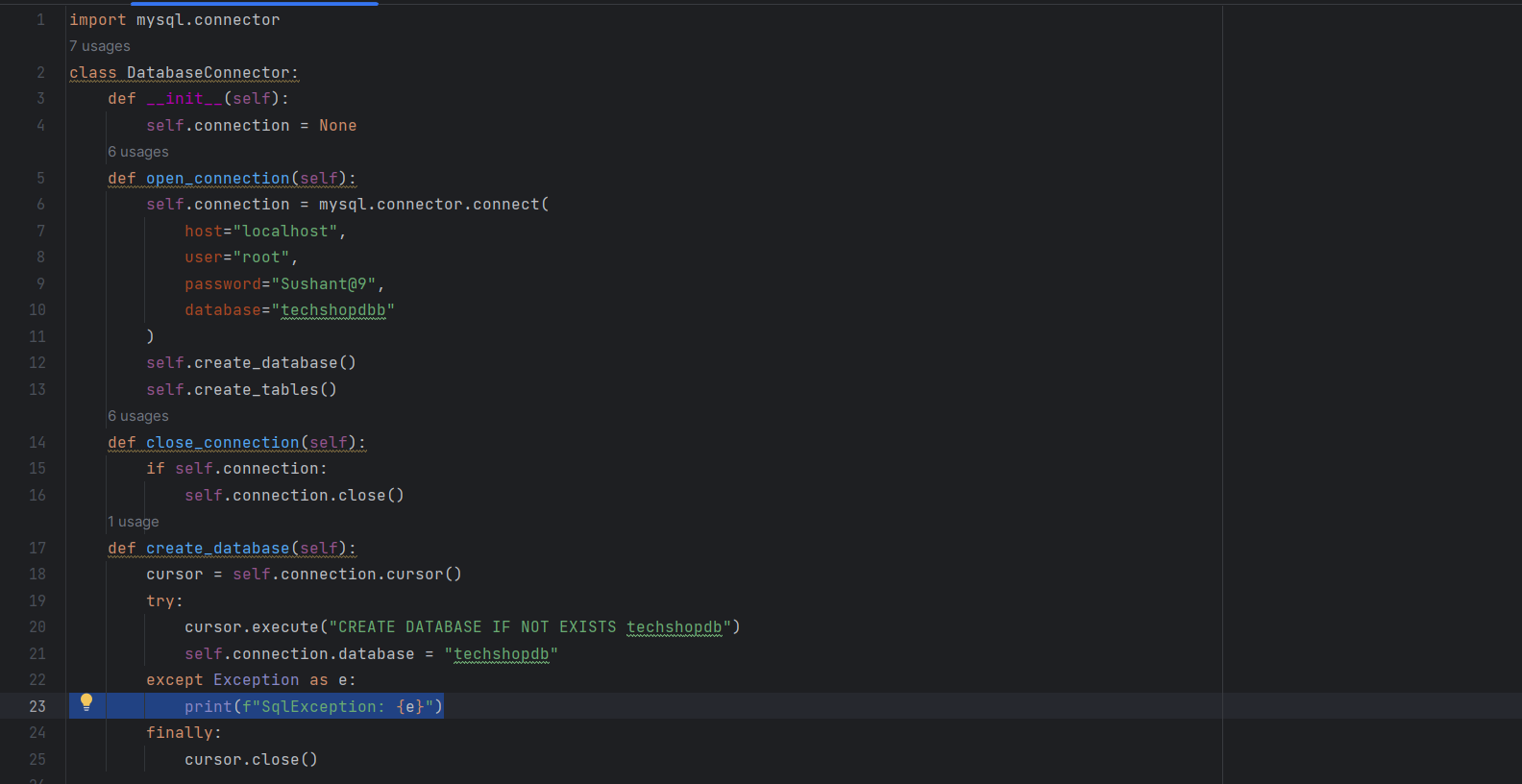


**• Database Access:**

o Challenge: Managing database connections and queries.

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

implement connection retries or failover mechanisms.



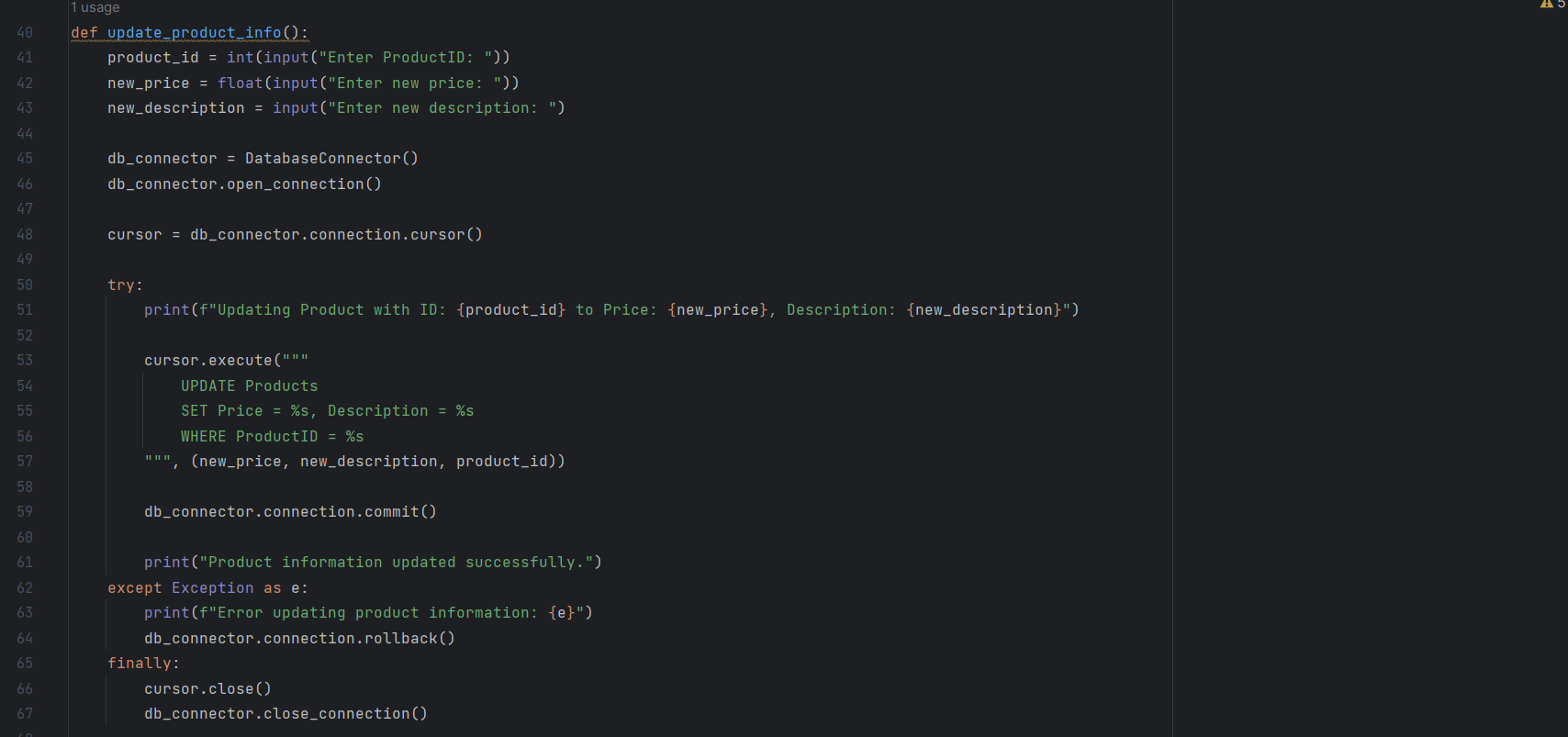
**Task 6: Collections**

**• Managing Products List:**

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

o Scenario: Adding, updating, products from the list.

o Solution: Implement methods to add, update, and remove products. Handle exceptions for duplicate products, invalid updates, or removal of products with existing orders.

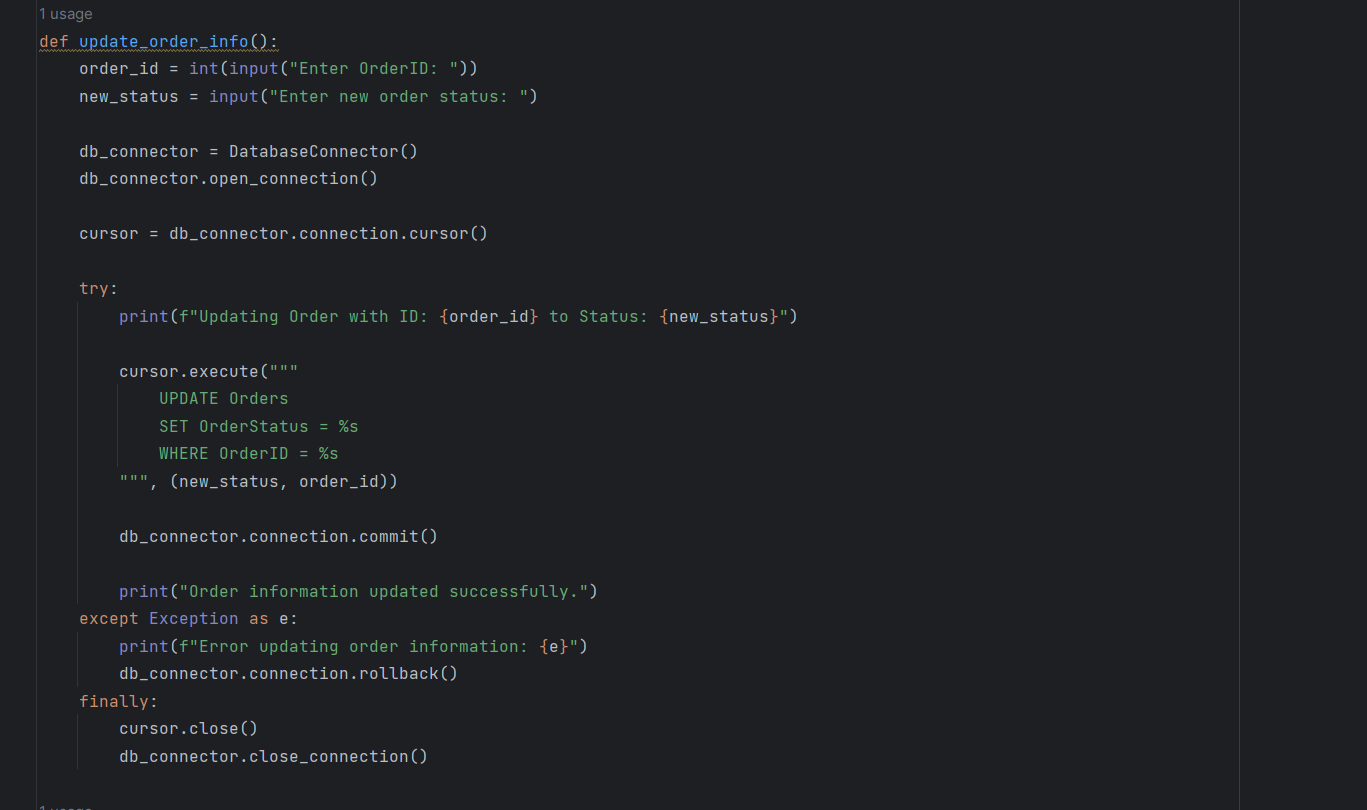


• **Managing Orders List:**

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

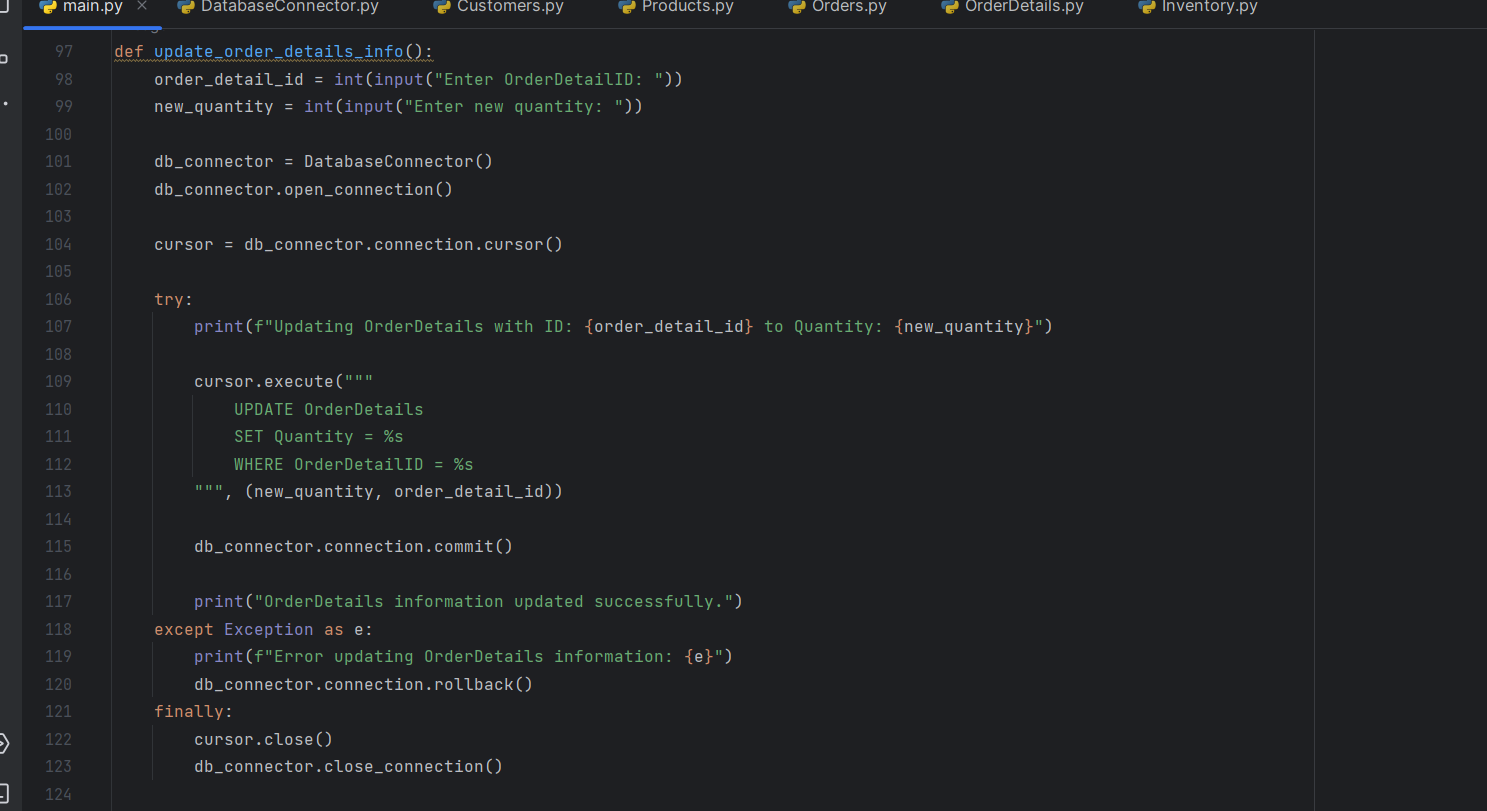
o Scenario: Adding new orders, updating order statuses, orders.

o Solution: Implement methods to add new orders, update order statuses, and remove canceled orders. Ensure that updates are synchronized with inventory and payment records.



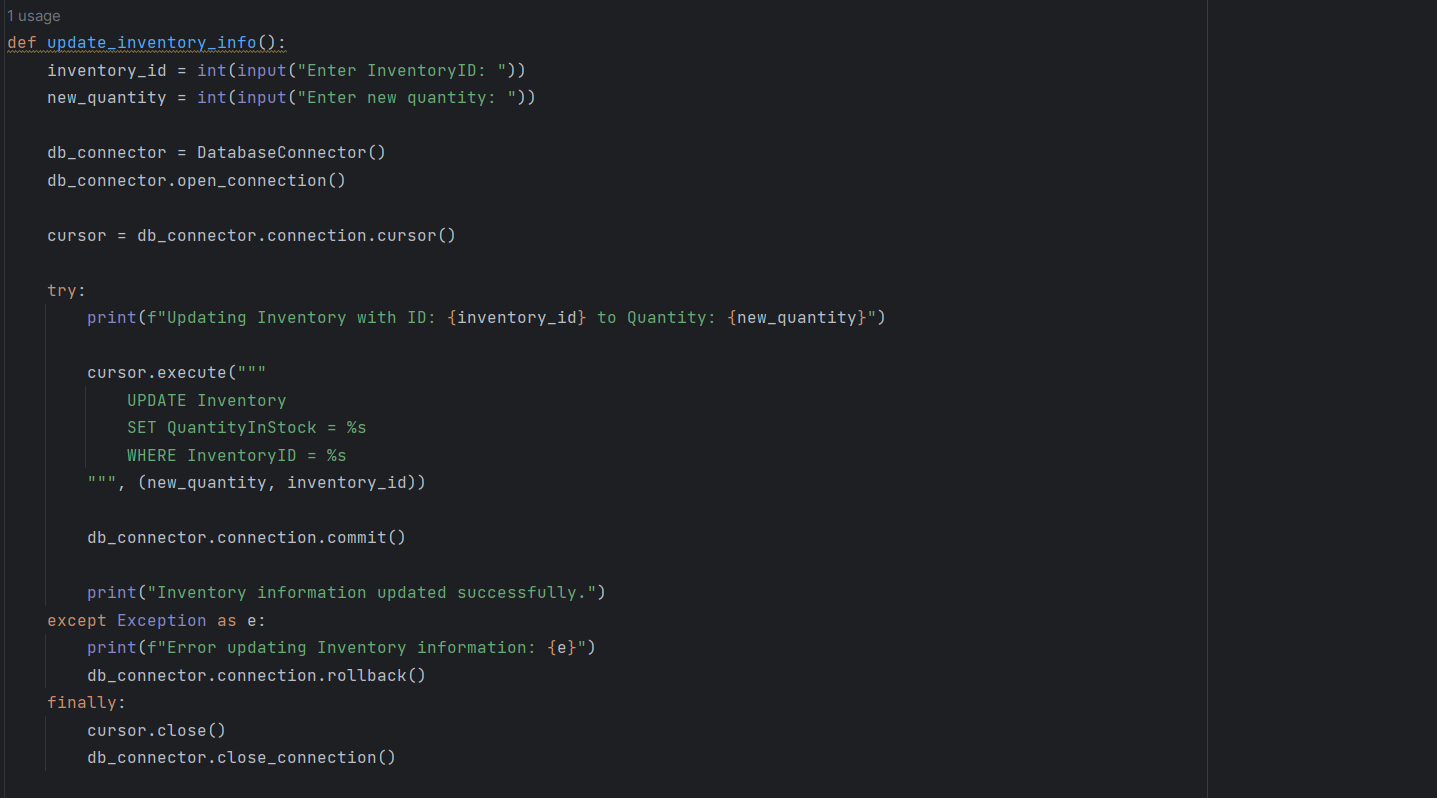
• **Sorting Orders by Date:**

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



• **Handling Inventory Updates:**

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



**Task 7: Database Connectivity**

• Implement a DatabaseConnector class responsible for establishing a connection to the

"TechShopDB" database. This class should include methods for opening, closing, and managing

database connections.

• Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties,

constructors, and methods for CRUD (Create, Read, Update, Delete) operations.

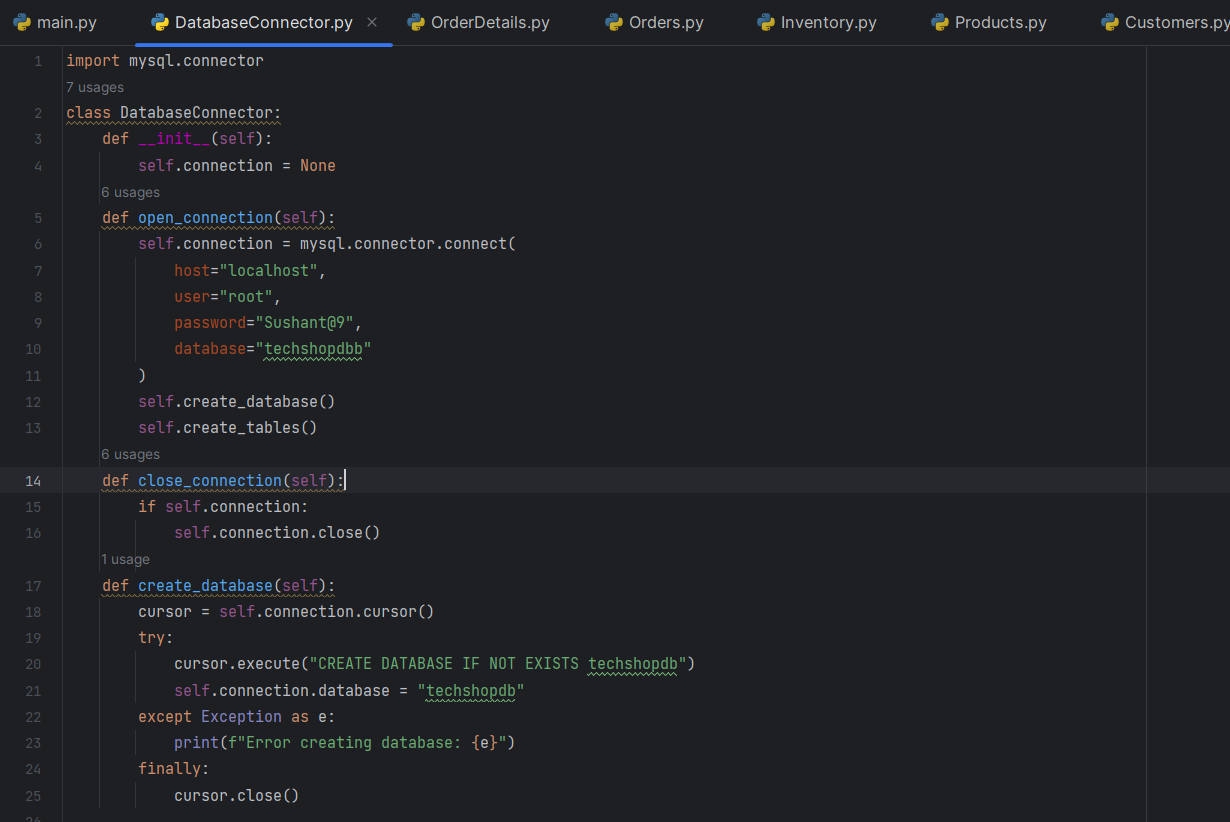
**1: Customer Registration**

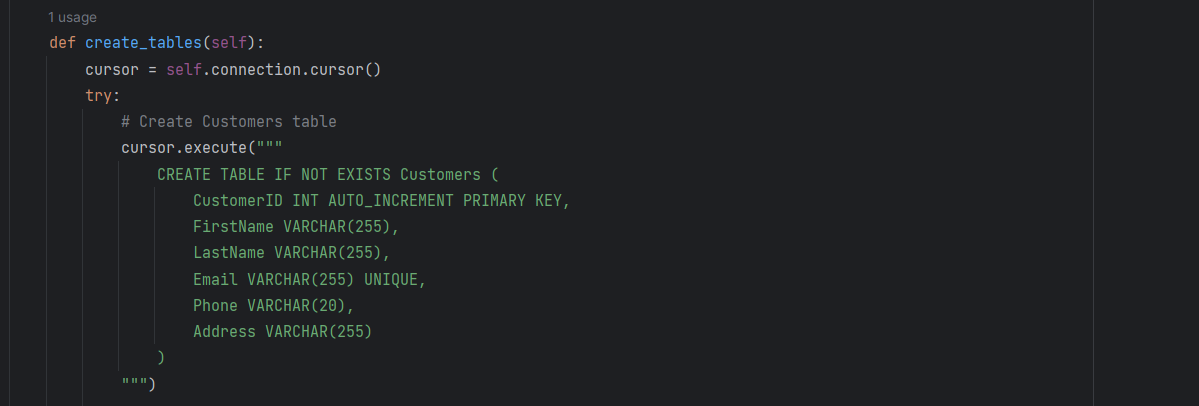
Description: When a new customer registers on the TechShop website, their information (e.g., name,

email, phone) needs to be stored in the database.

Task: Implement a registration form and database connectivity to insert new customer records. Ensure

proper data validation and error handling for duplicate email addresses.



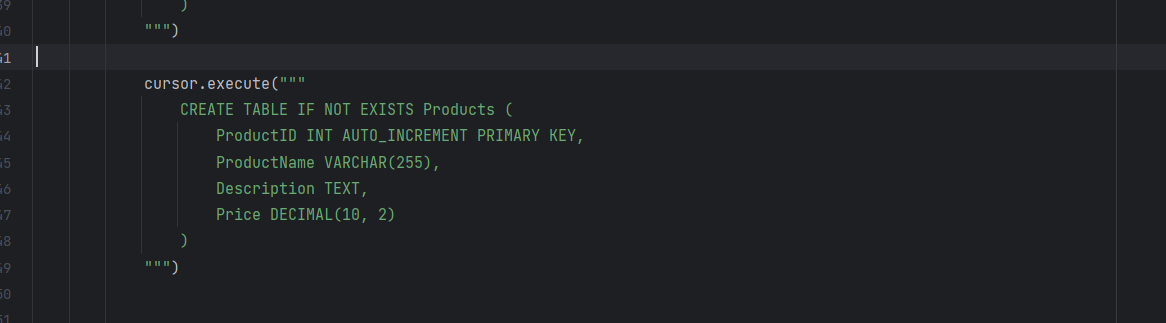


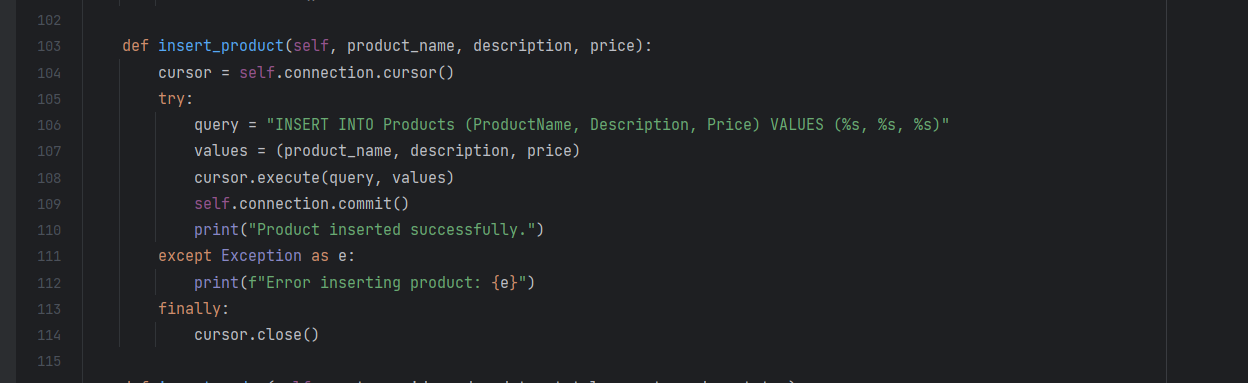


**2: Product Catalog Management**

Description: TechShop regularly updates its product catalog with new items and changes in product

details (e.g., price, description). These changes need to be reflected in the database.





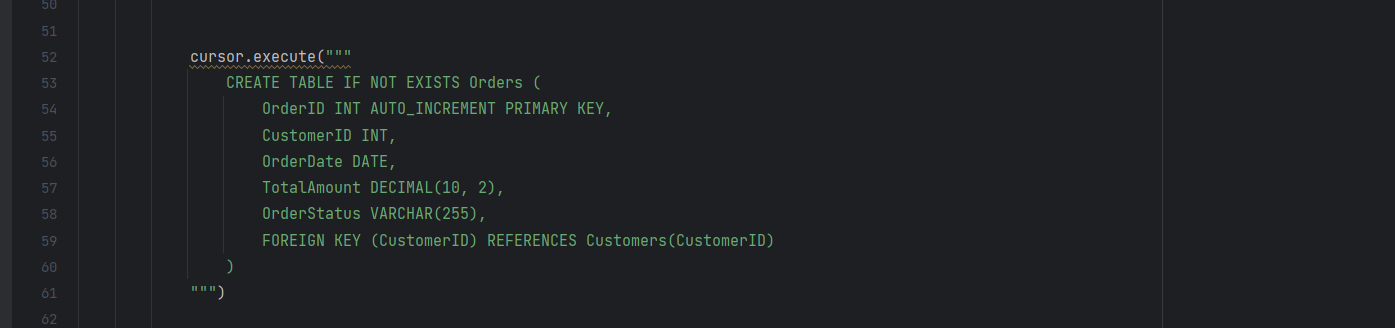
**3: Placing Customer Orders**

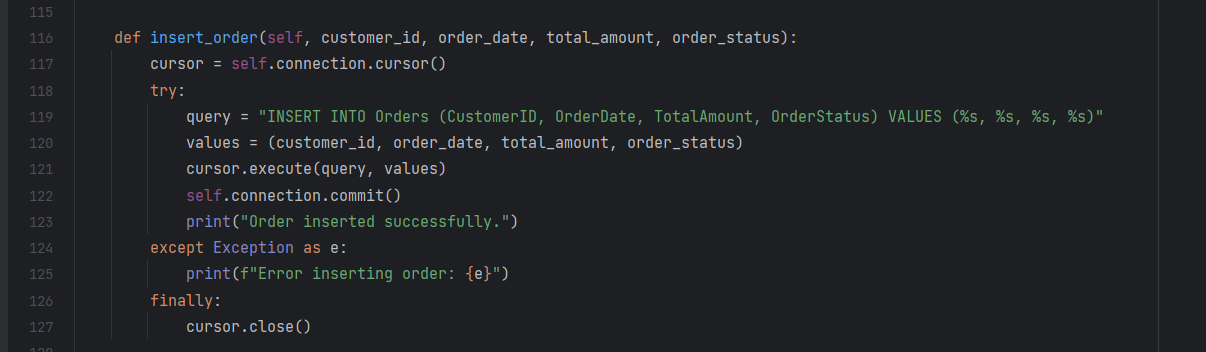
Description: Customers browse the product catalog and place orders for products they want to

purchase. The orders need to be stored in the database.

Task: Implement an order processing system. Use database connectivity to record customer orders,

.

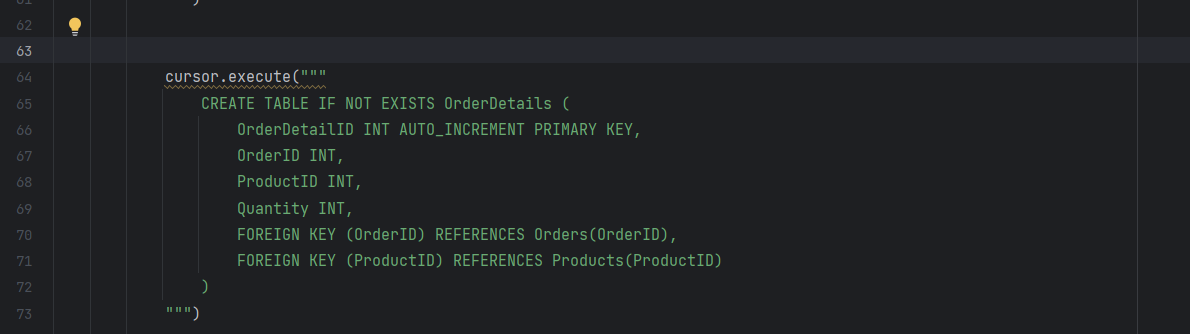


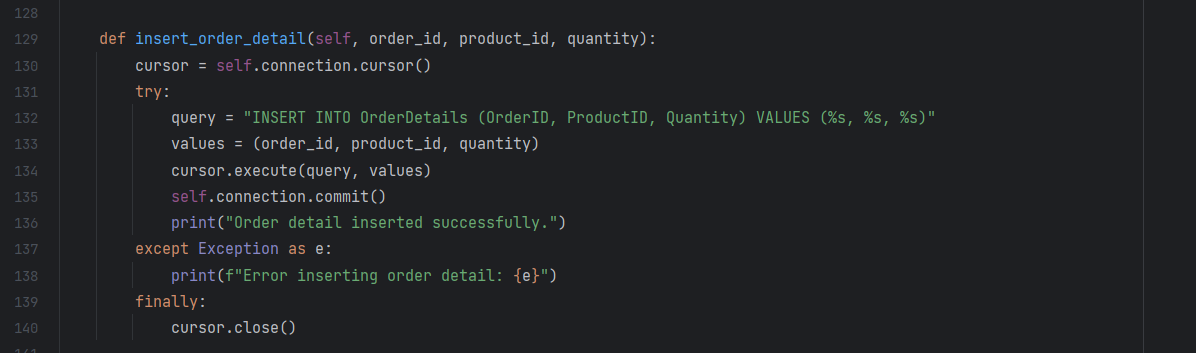


**4: Tracking Order Status**

Description: Customers and employees need to track the status of their orders. The order status

information is stored in the database.



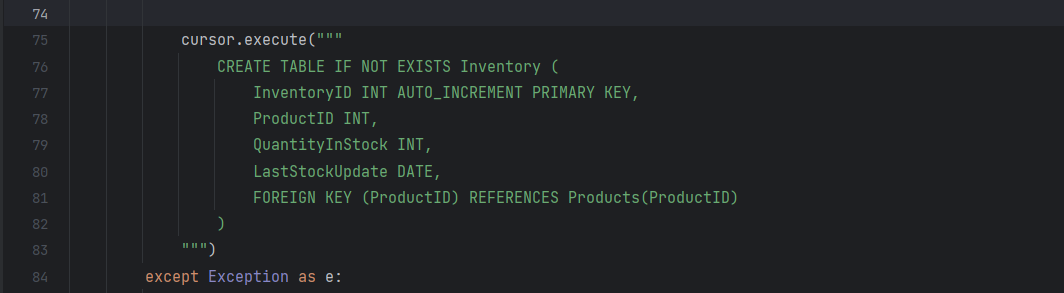


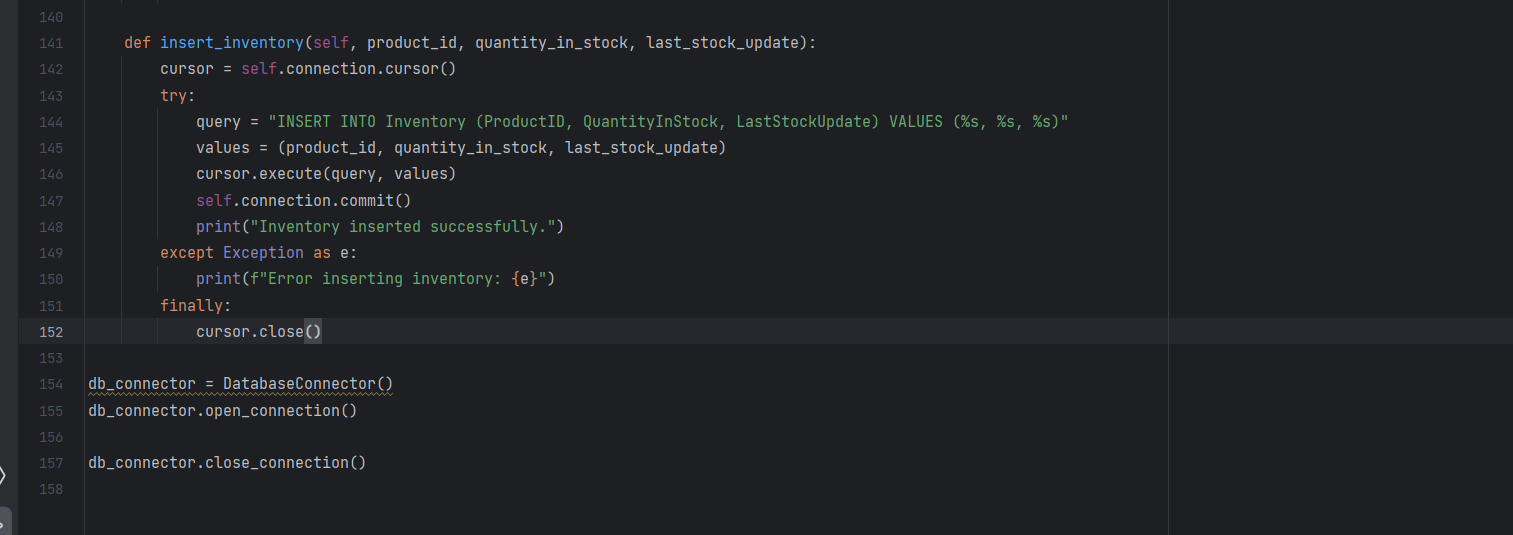
**5: Inventory Management**

Description: TechShop needs to manage product inventory, including adding new products, updating

stock levels, and removing discontinued items.

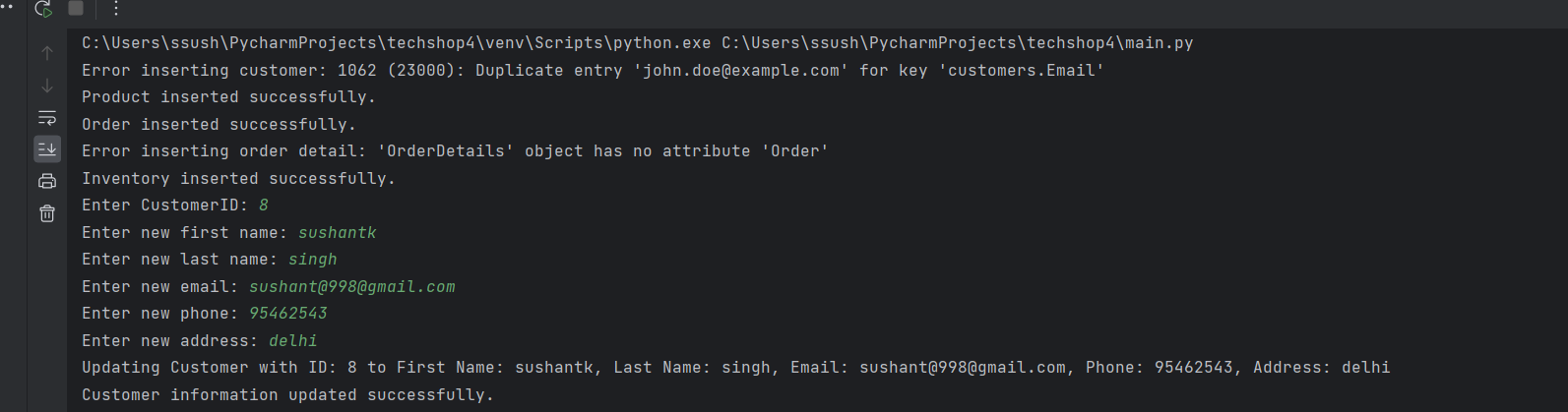
Task: Create an inventory management system with database connectivity.

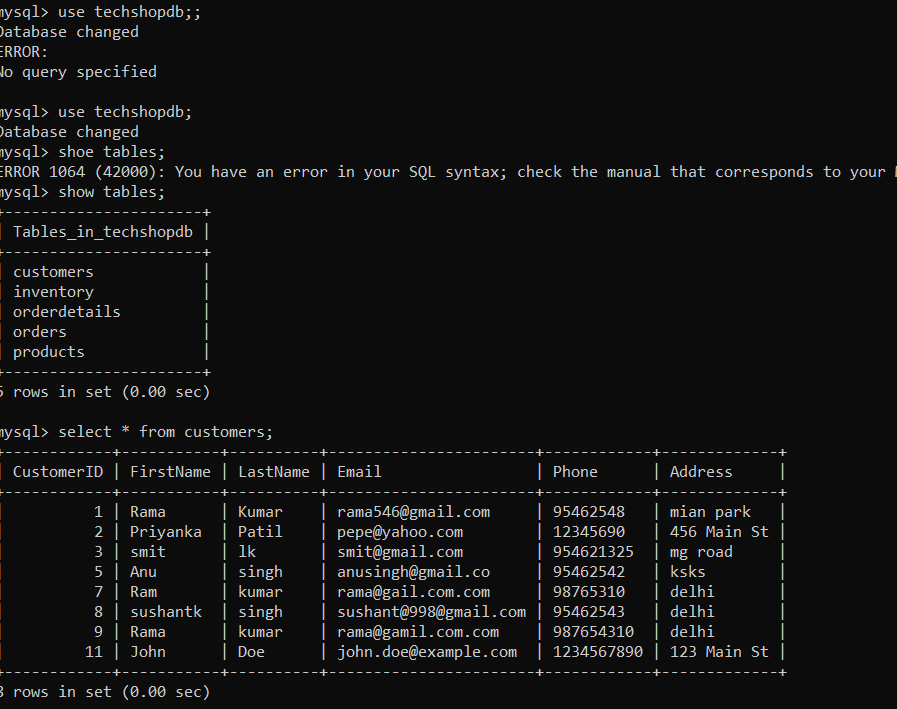




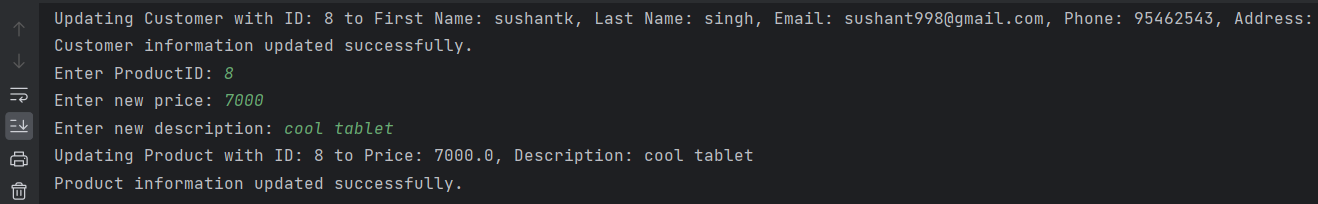
**OUTPUTS (** Showing the output of each given case according to the question’s )

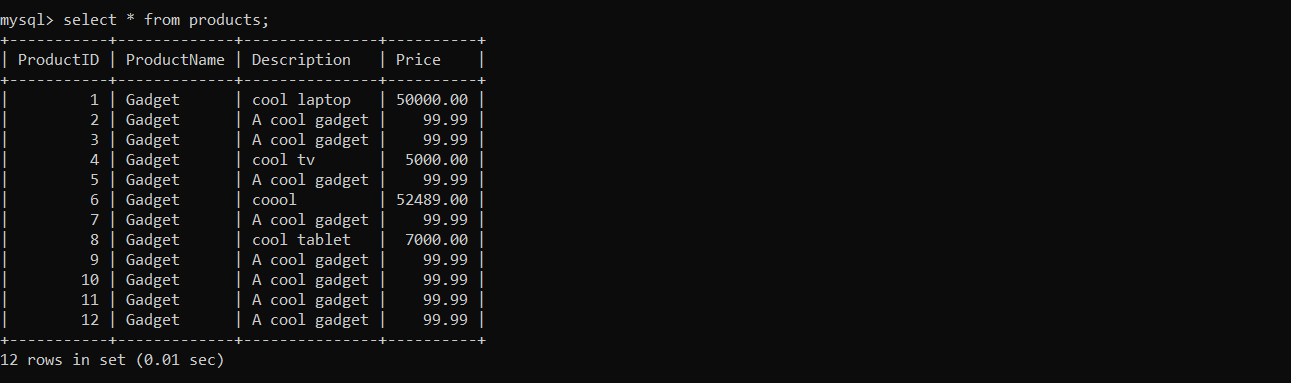
**Customer Input/Output**



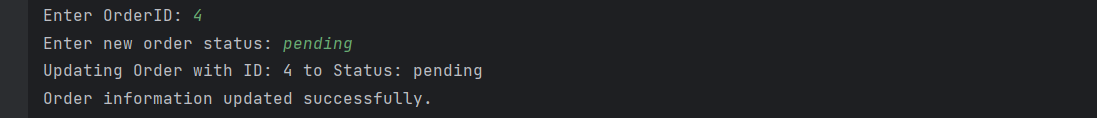


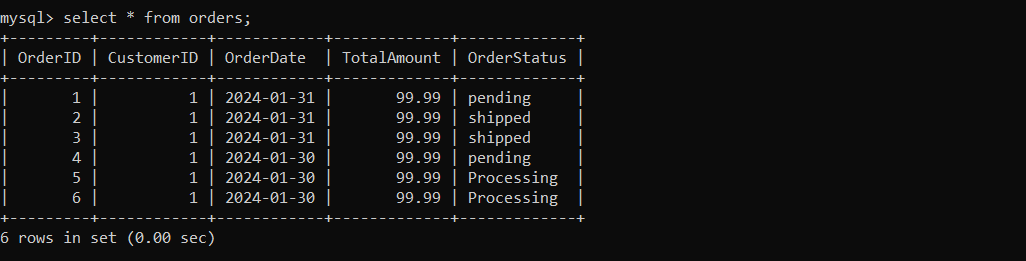
**Products Input/Output**





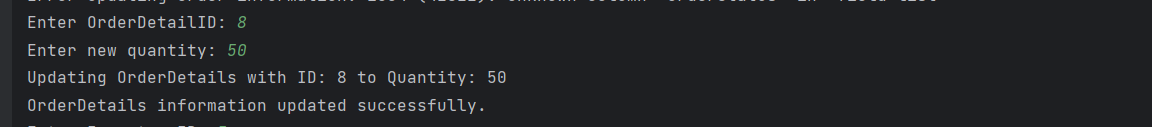
**Orders Input/Output**

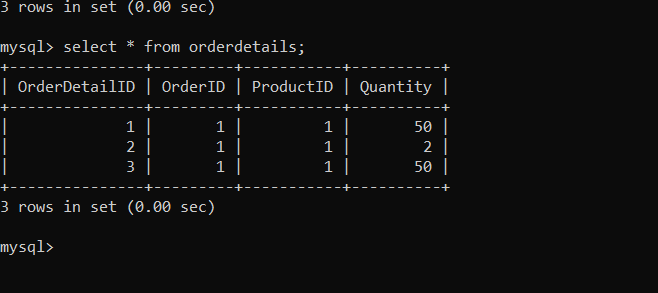




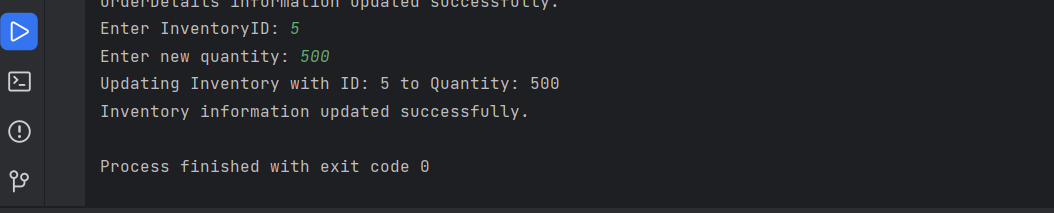
**OrderDetails Input/Output**

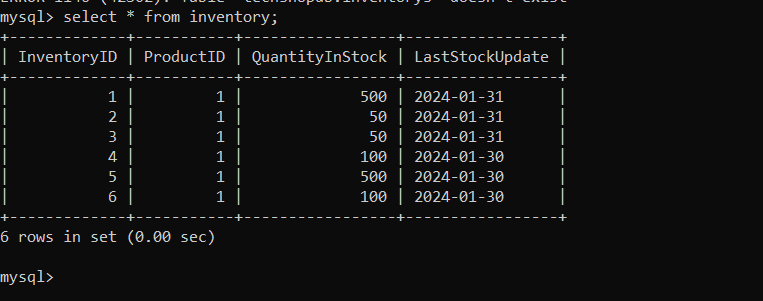
.





**Inventory Input/Output**





**\*\*\*\*\*\*\*\* ThankYou \*\*\*\*\*\*\*\***