**Name: Sarthak Shandilya**

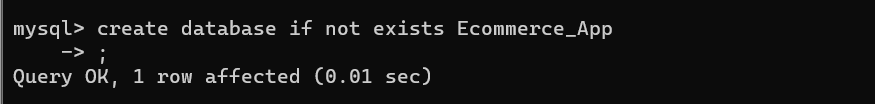
**Case study 6 (Ecommerce)**

**Submitted to: Karthika**

**Hexaware Technologies**

**Schema Design:**

**Create database Ecommerce.**



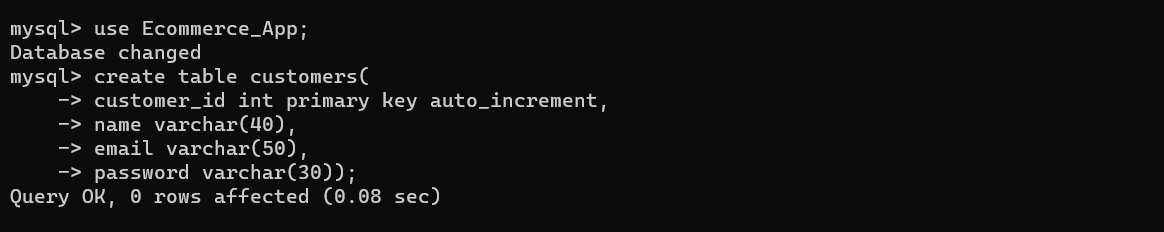
1. customers table:

• customer\_id (Primary Key)

• name

• email

• password



1. products table:

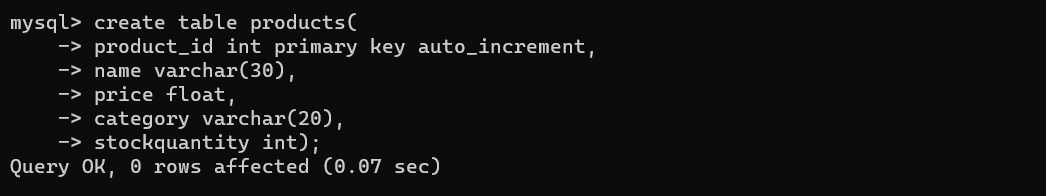
• product\_id (Primary Key)

• name

• price

• description

• stockQuantity



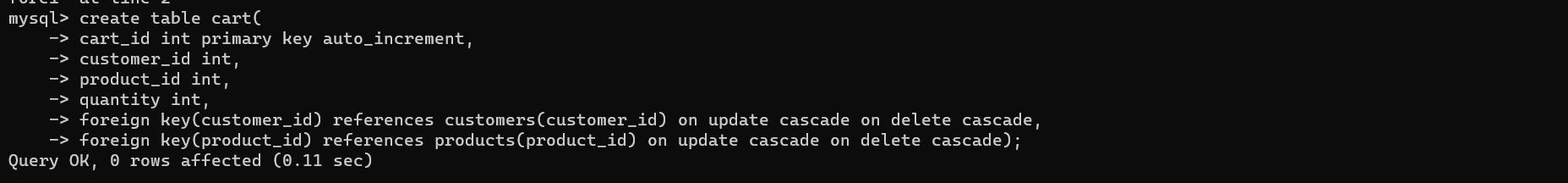
1. cart table:

• cart\_id (Primary Key)

• customer\_id (Foreign Key)

• product\_id (Foreign Key)

• quantity



1. orders table:

• order\_id (Primary Key)

• customer\_id (Foreign Key)

• order\_date

• total\_price

• shipping\_address



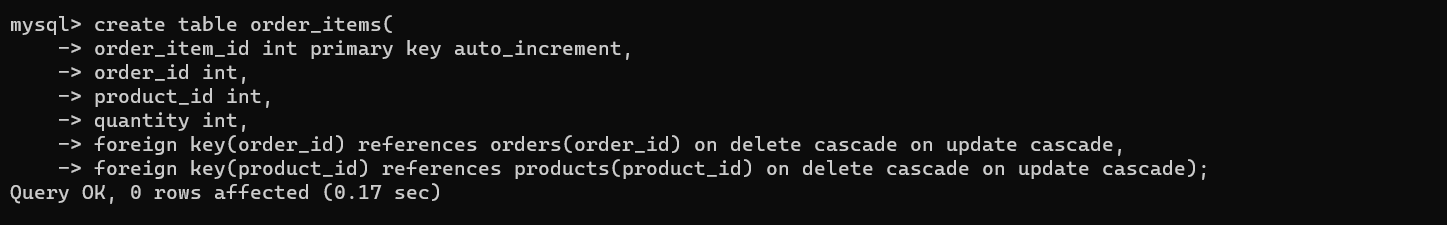
1. order\_items table (to store order details):

• order\_item\_id (Primary Key)

• order\_id (Foreign Key)

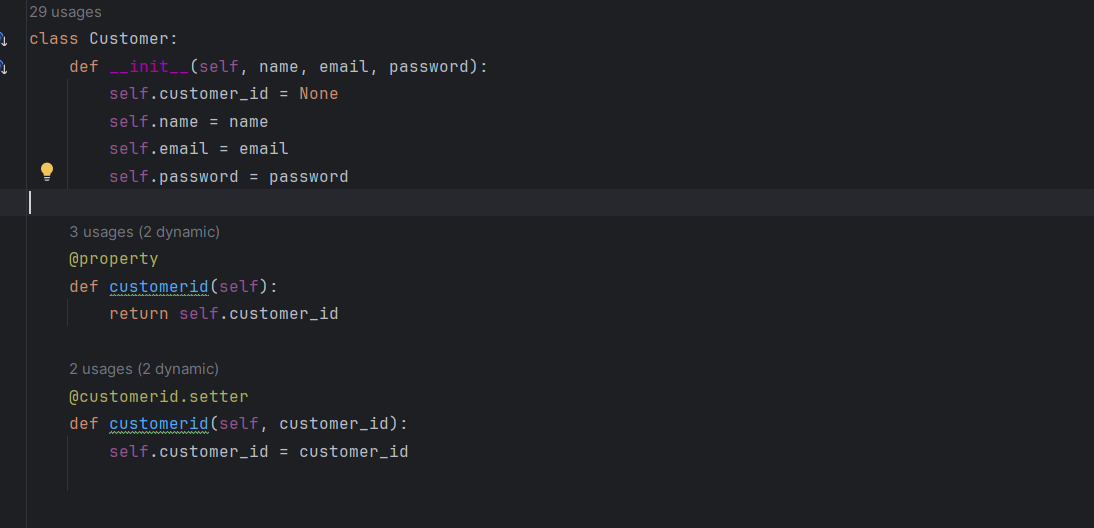
• product\_id (Foreign Key)

• quantity



**Classes:**

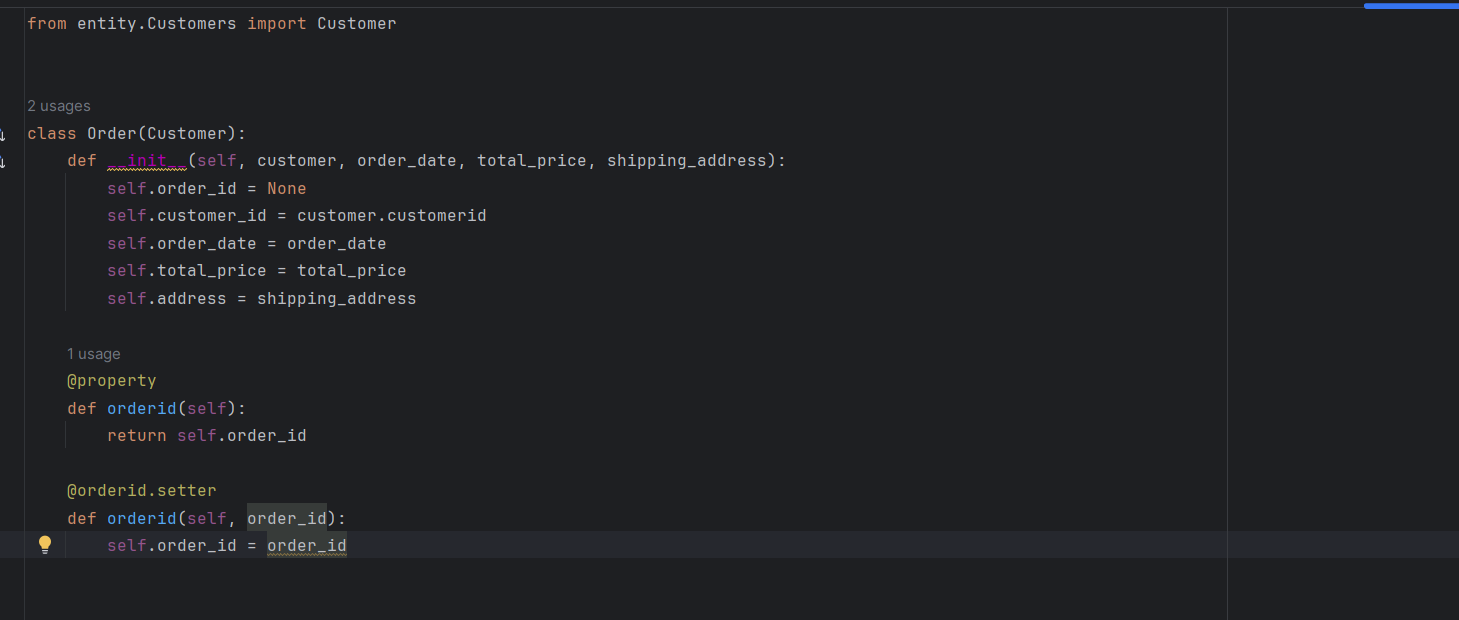
**Customer:**



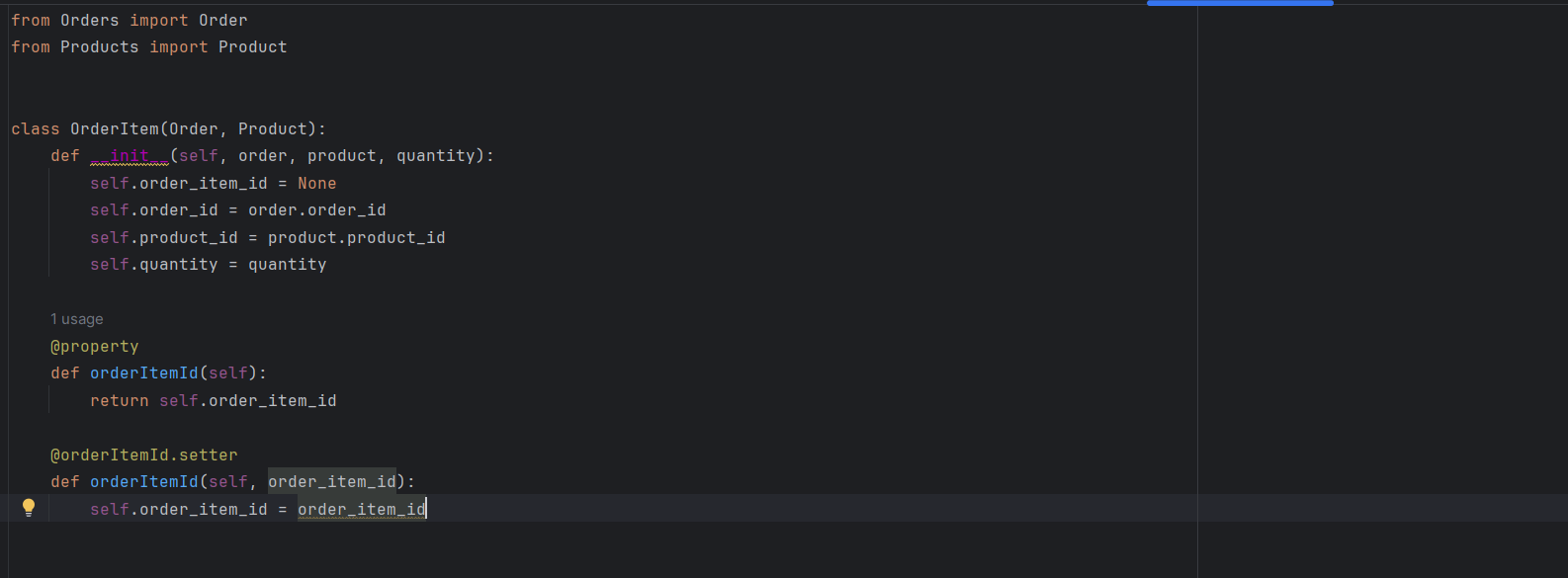
**Products:**

****

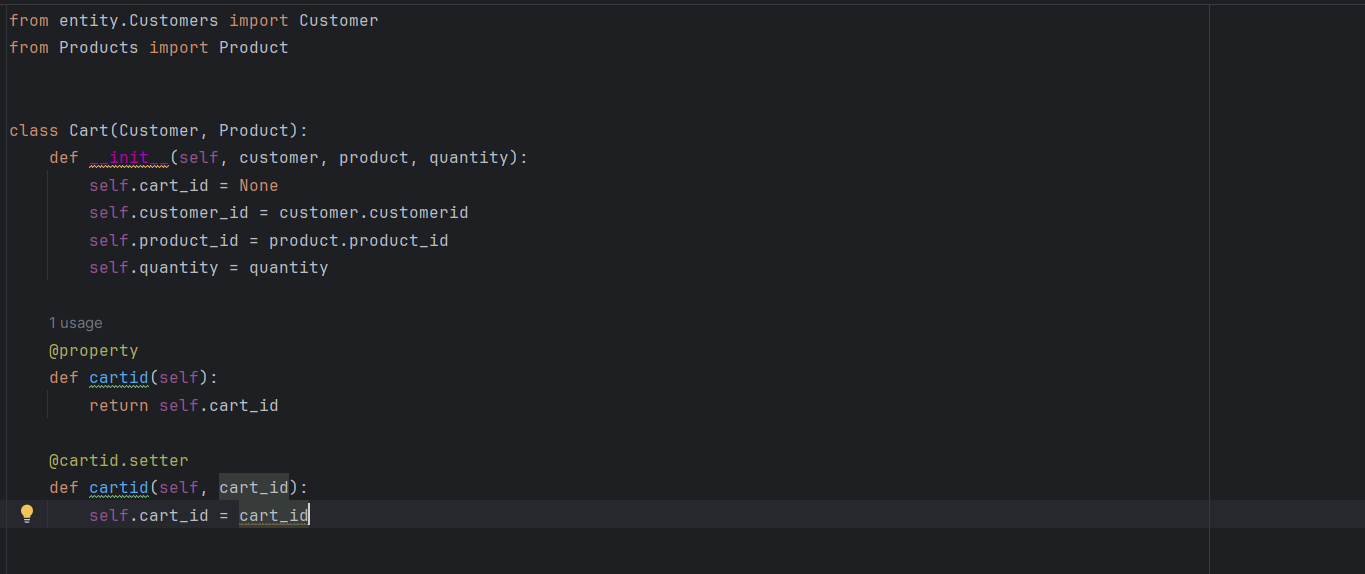
**Orders:**

****

**OrderItems:**

****

**Cart:**

****

**Service Provider Interface/Abstract class:**

**Keep the interfaces and implementation classes in package dao**

• Define an OrderProcessorRepository interface/abstract class with methods for adding/removing products to/from the cart and placing orders.

The following methods will interact with database.

1. createProduct()

parameter: Product product return type: boolean

2. createCustomer()

parameter: Customer customer

return type: boolean

3. deleteProduct() parameter: productId return type: boolean

4. deleteCustomer(customerId) parameter: customerId return type: boolean

5. addToCart(): insert the product in cart. parameter: Customer customer, Product product, int quantity

return type: boolean

6. removeFromCart(): delete the product in cart. parameter: Customer customer, Product product

return type: boolean

7. getAllFromCart(Customer customer): list the product in cart for a customer.

parameter: Customer customer

return type: list of product

8. placeOrder(Customer customer, List<Map>, string shippingAddress): should update order table and orderItems table.

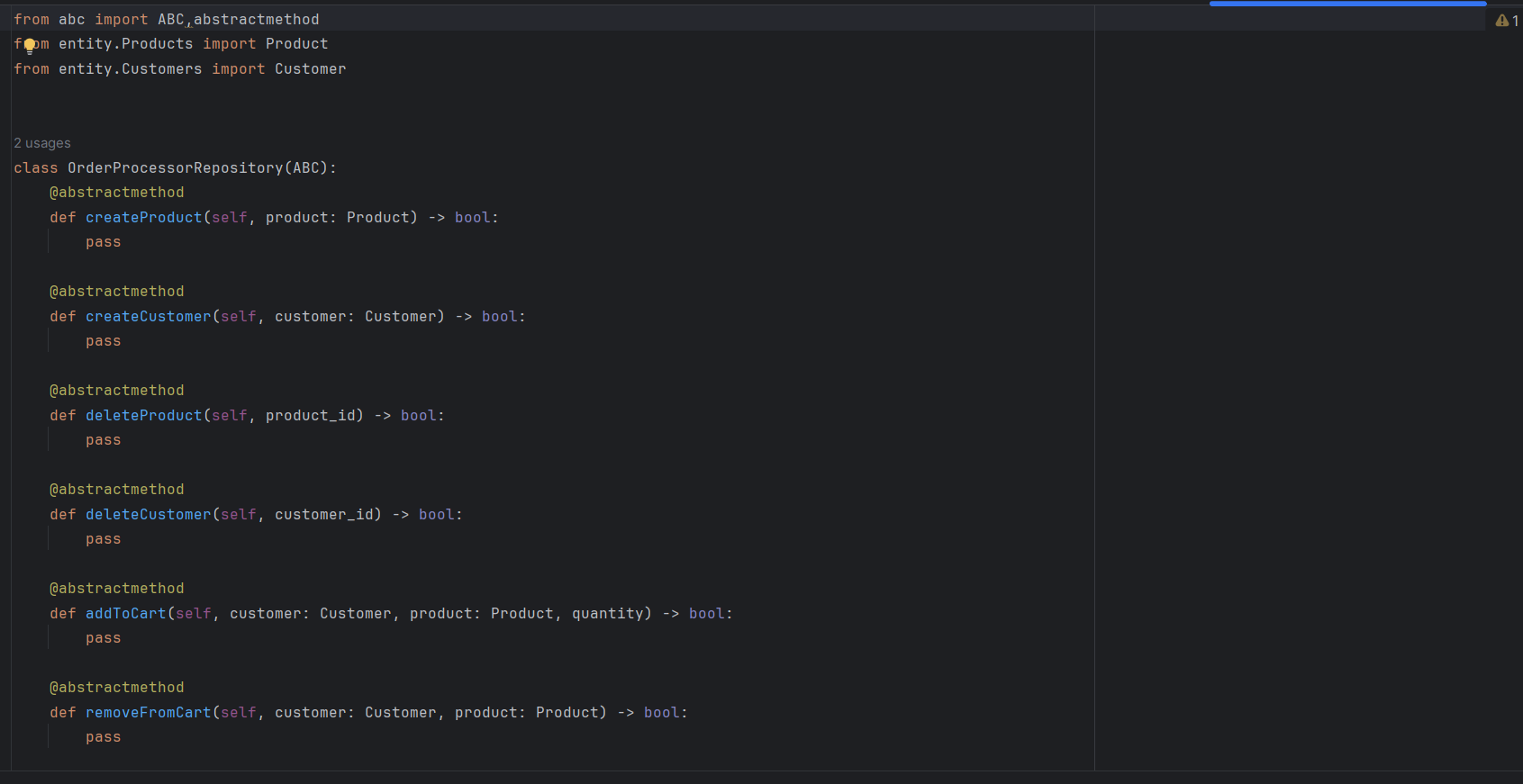
parameter: Customer customer, list of product and quantity

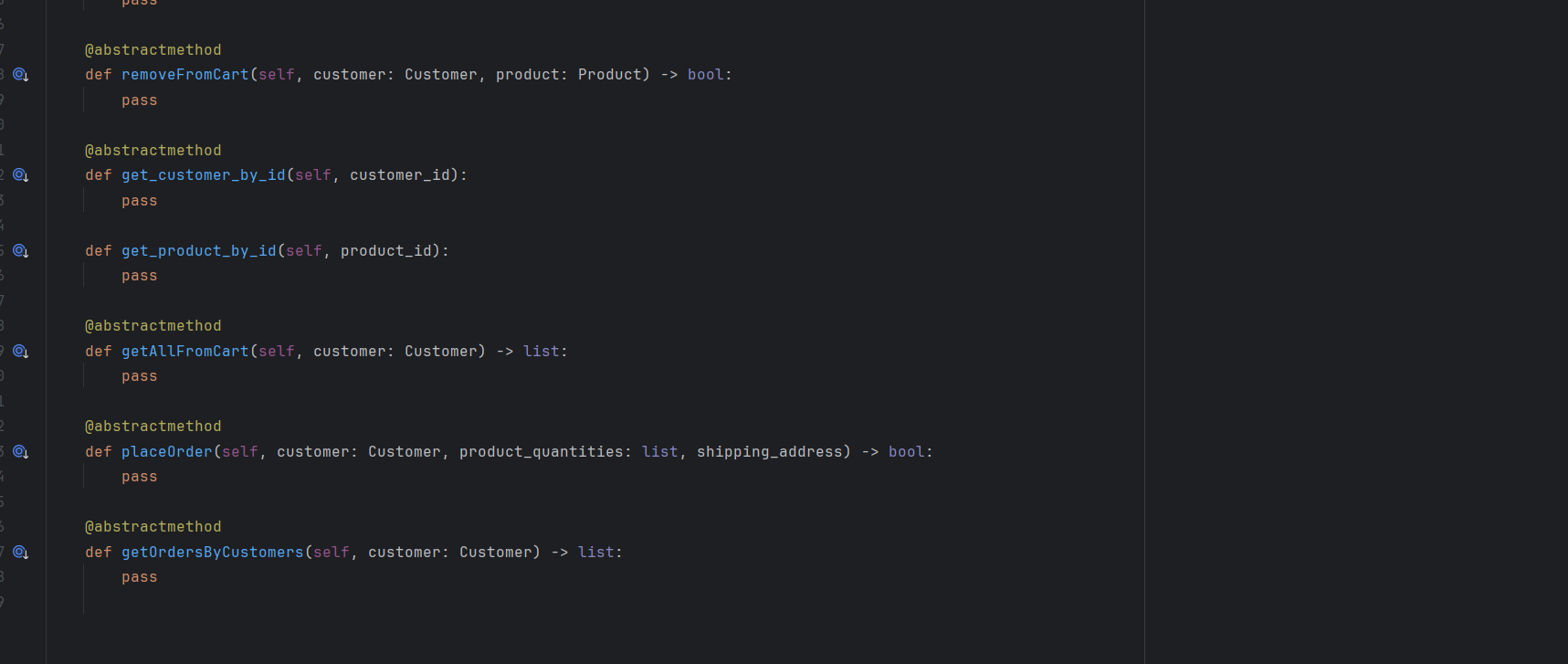
return type: boolean

9. getOrdersByCustomer()

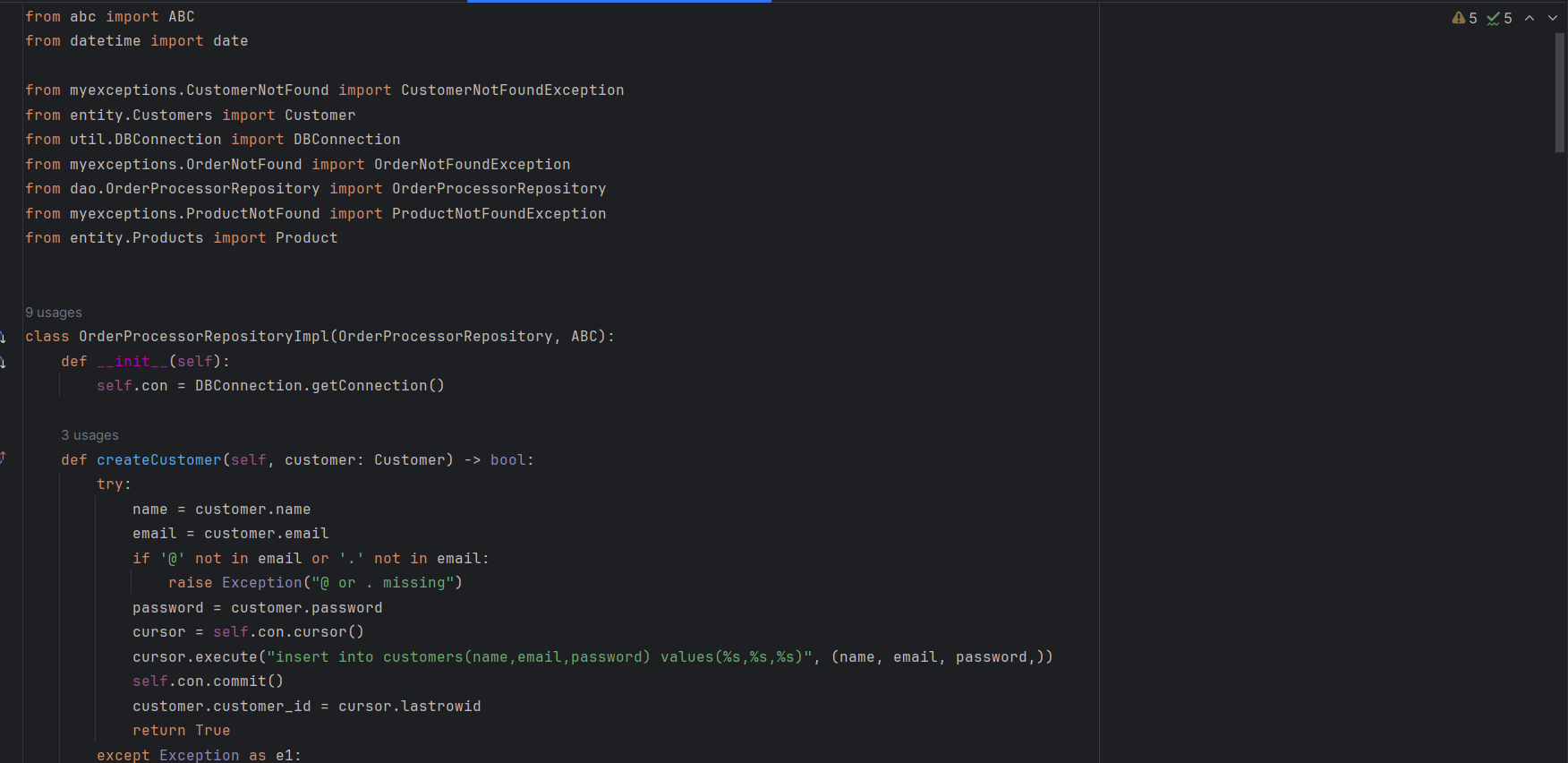
parameter: customerid

return type: list of product and quantity

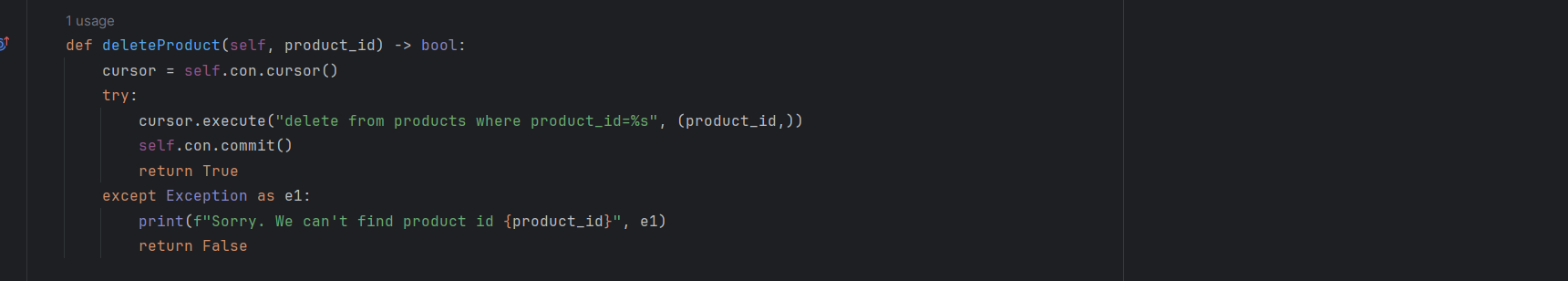


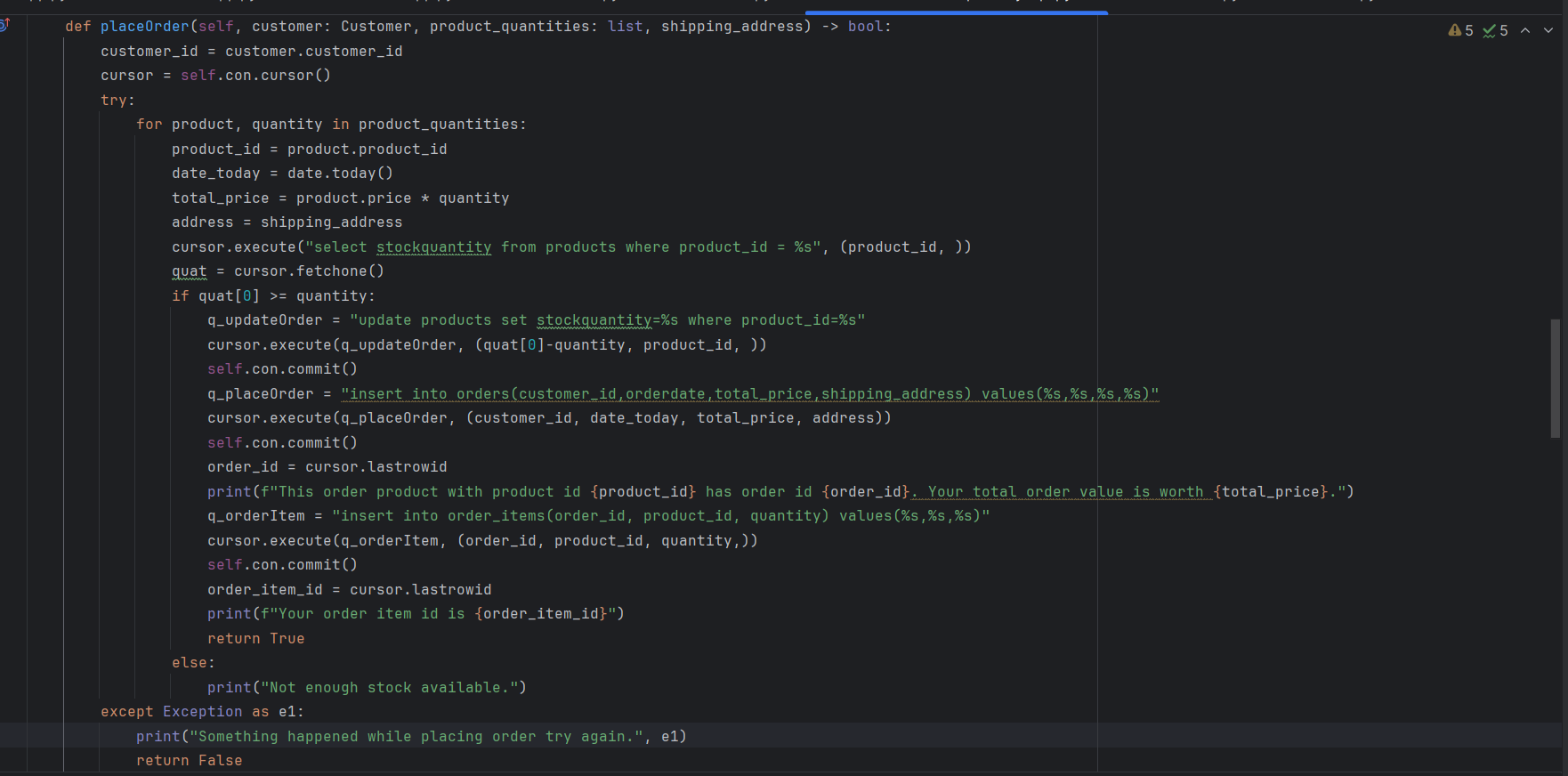


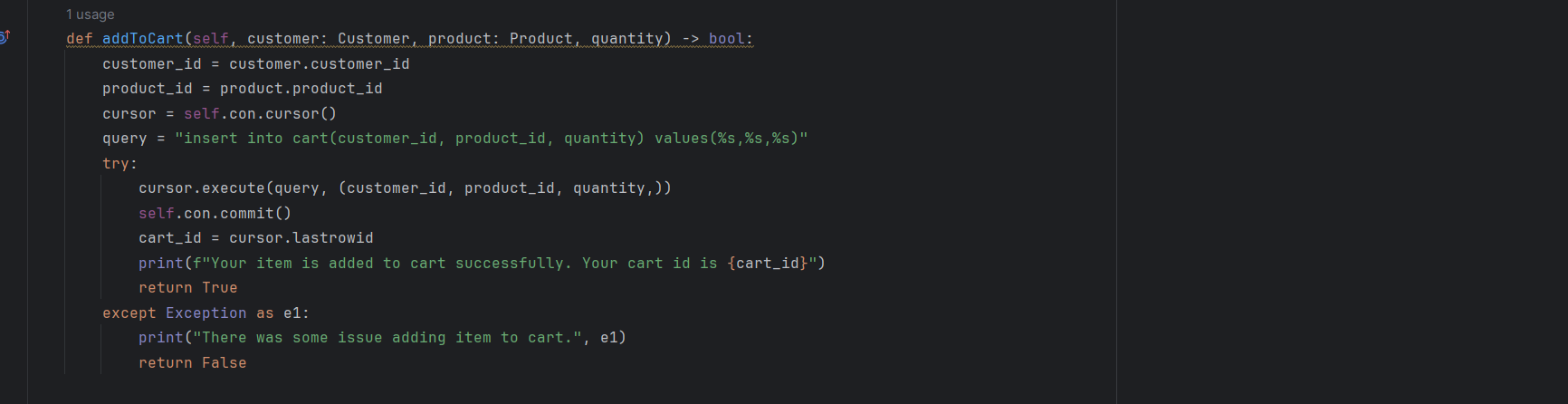
**Implement the above interface in a class called OrderProcessorRepositoryImpl in package dao.**

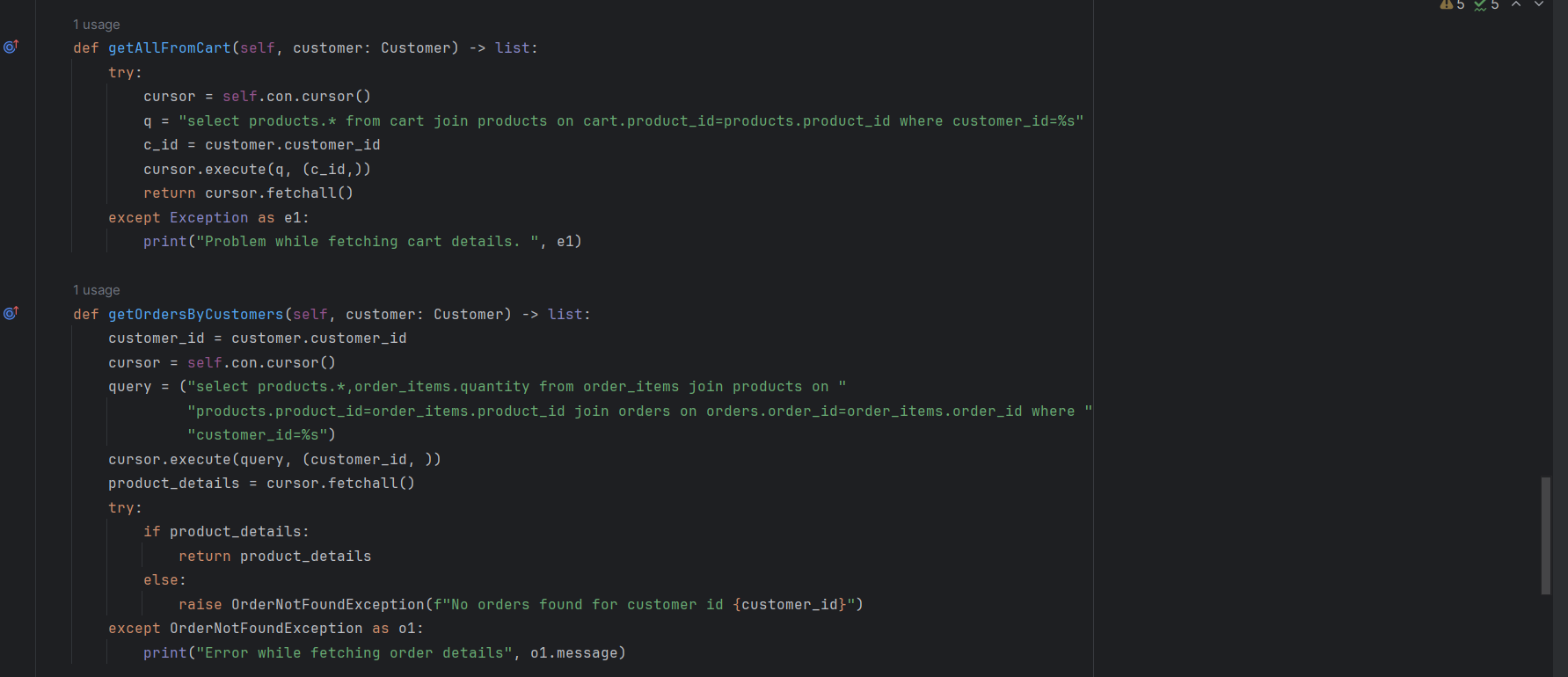
****

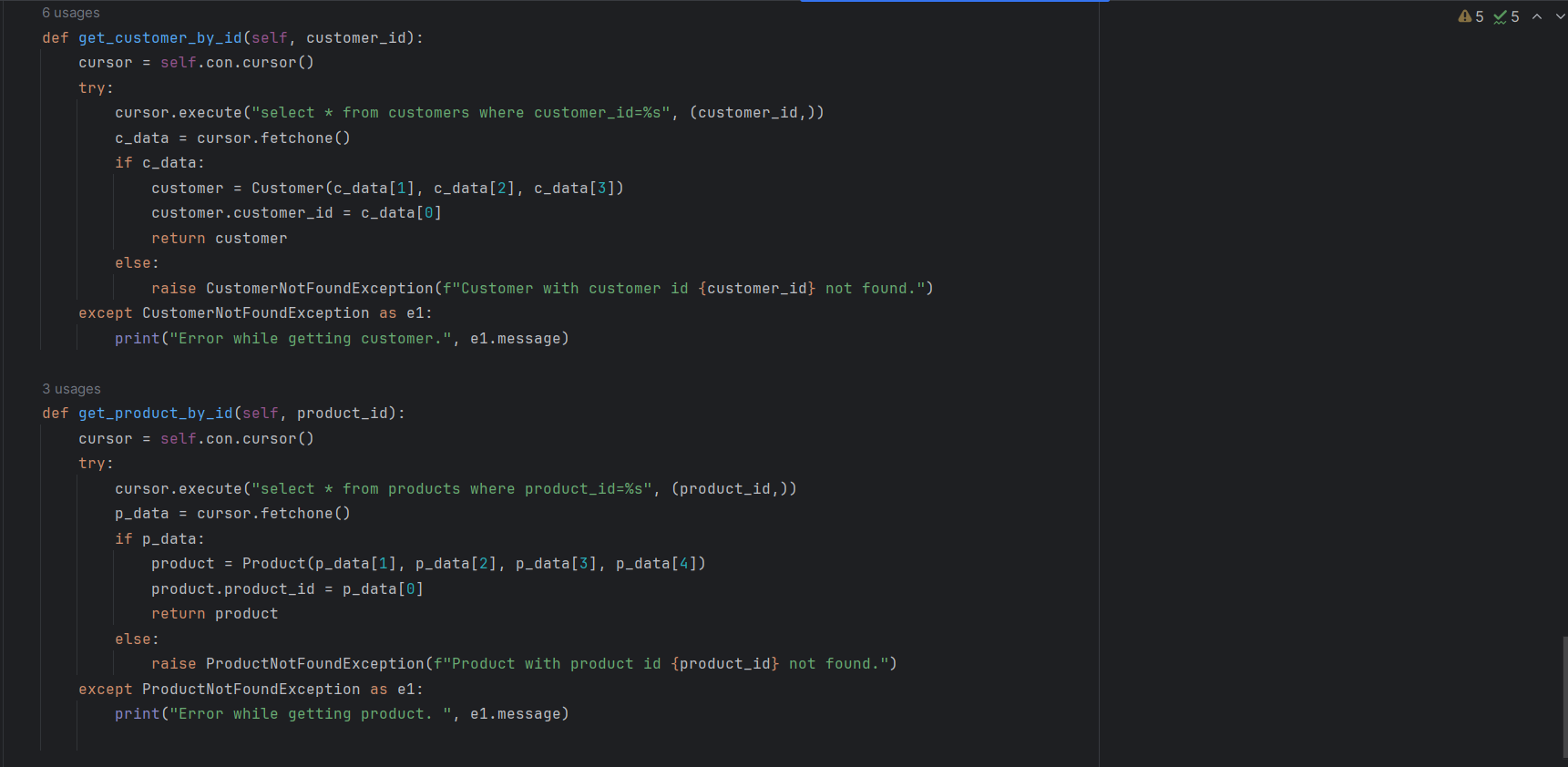
****

****

****

****

****

****

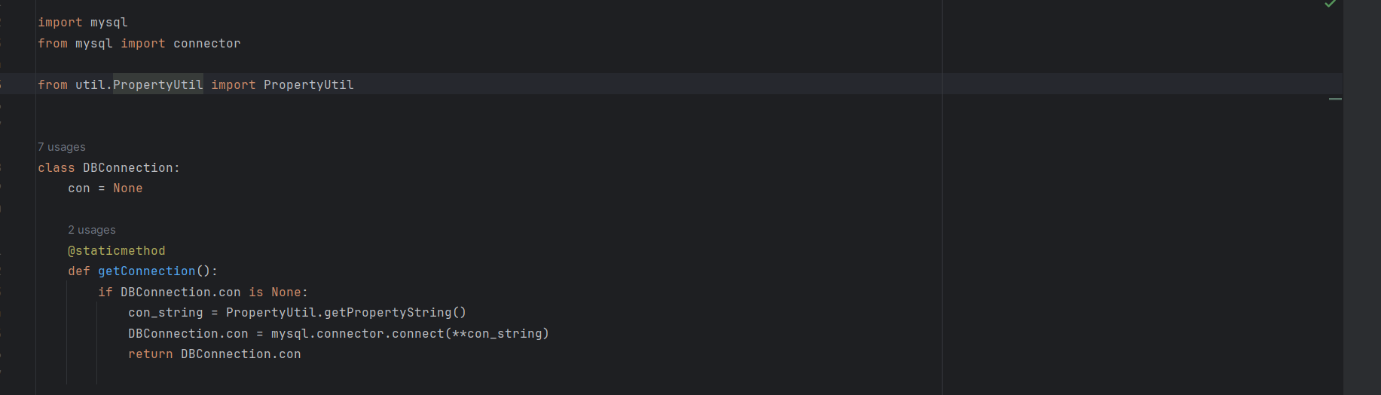
**These are the implementations of every abstract methods.**

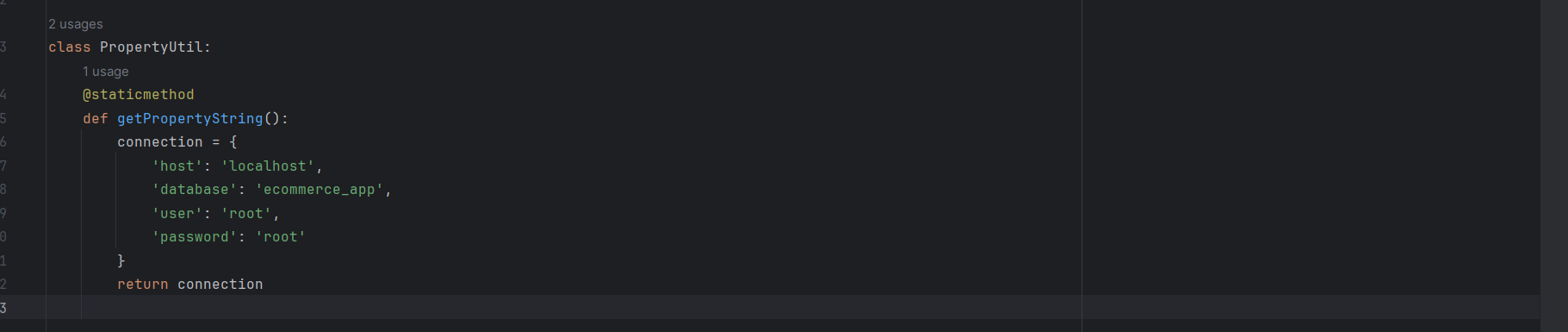
**Write code to establish a connection to your SQL database.**

• Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.

• Connection properties supplied in the connection string should be read from a property file.

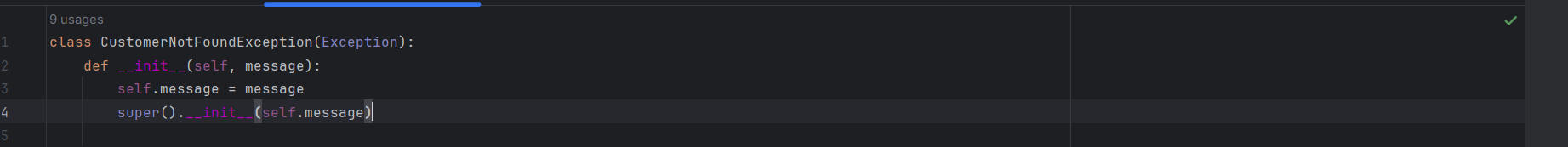
• Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

****

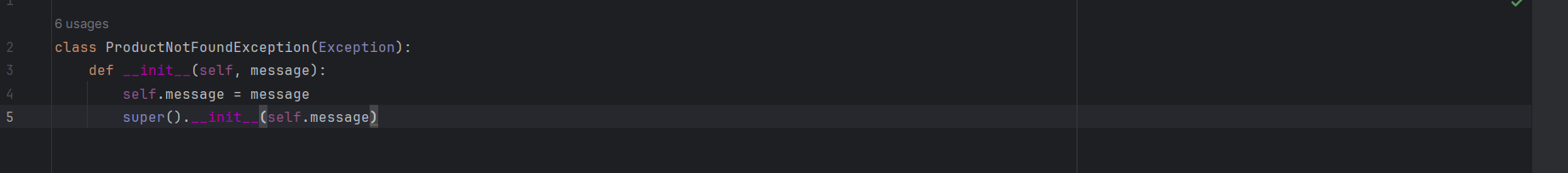
****

Create the exceptions in package myexceptions and create the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method

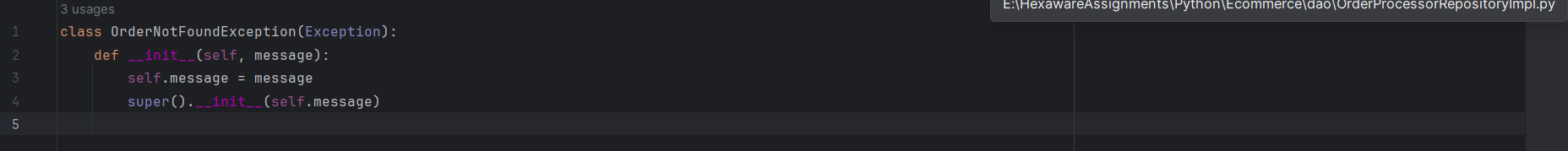
• CustomerNotFoundException: throw this exception when user enters an invalid customer id which doesn’t exist in db



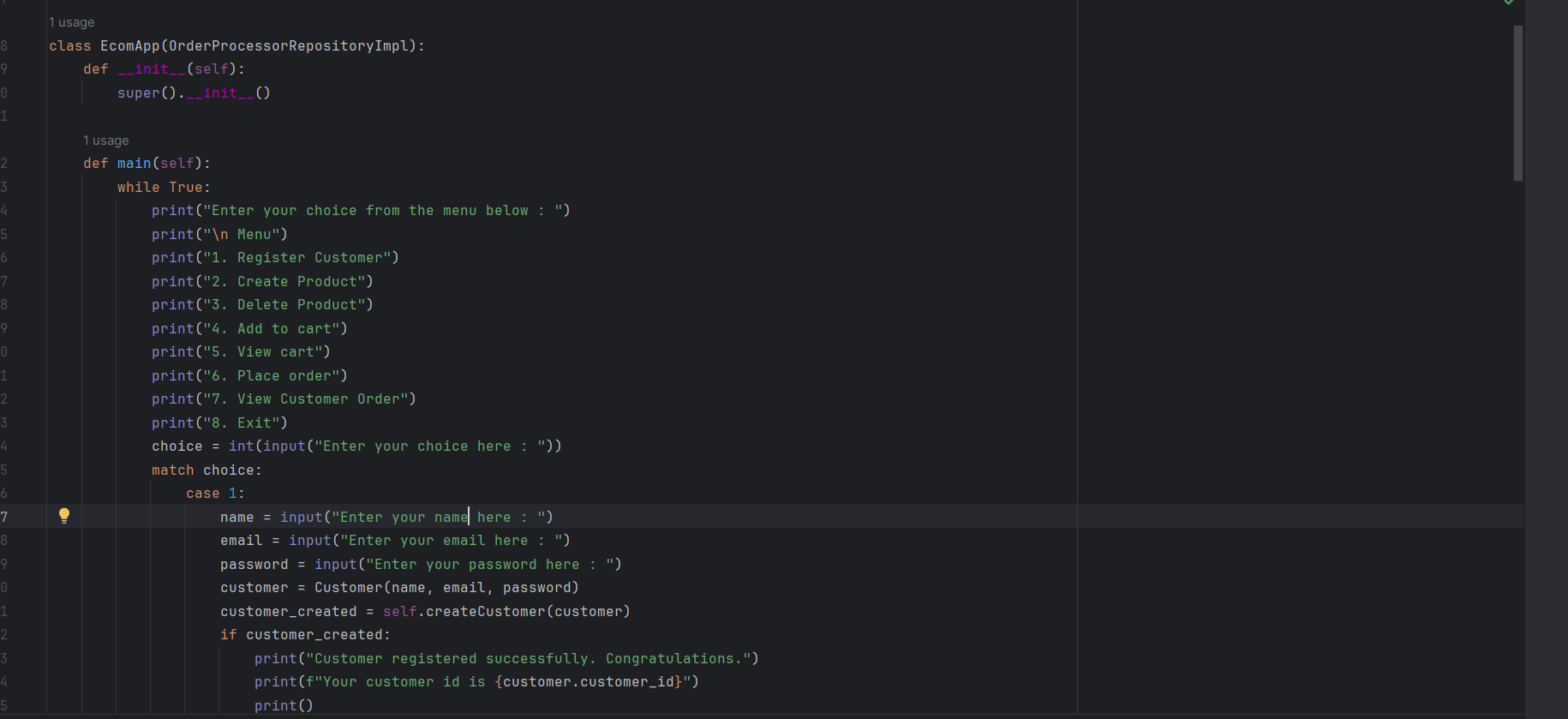
• ProductNotFoundException: throw this exception when user enters an invalid product id which doesn’t exist in db



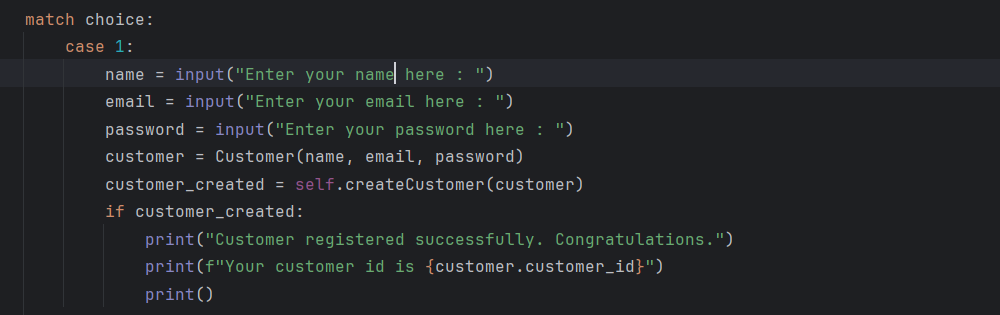
• OrderNotFoundException: throw this exception when user enters an invalid order id which doesn’t exist in db

****

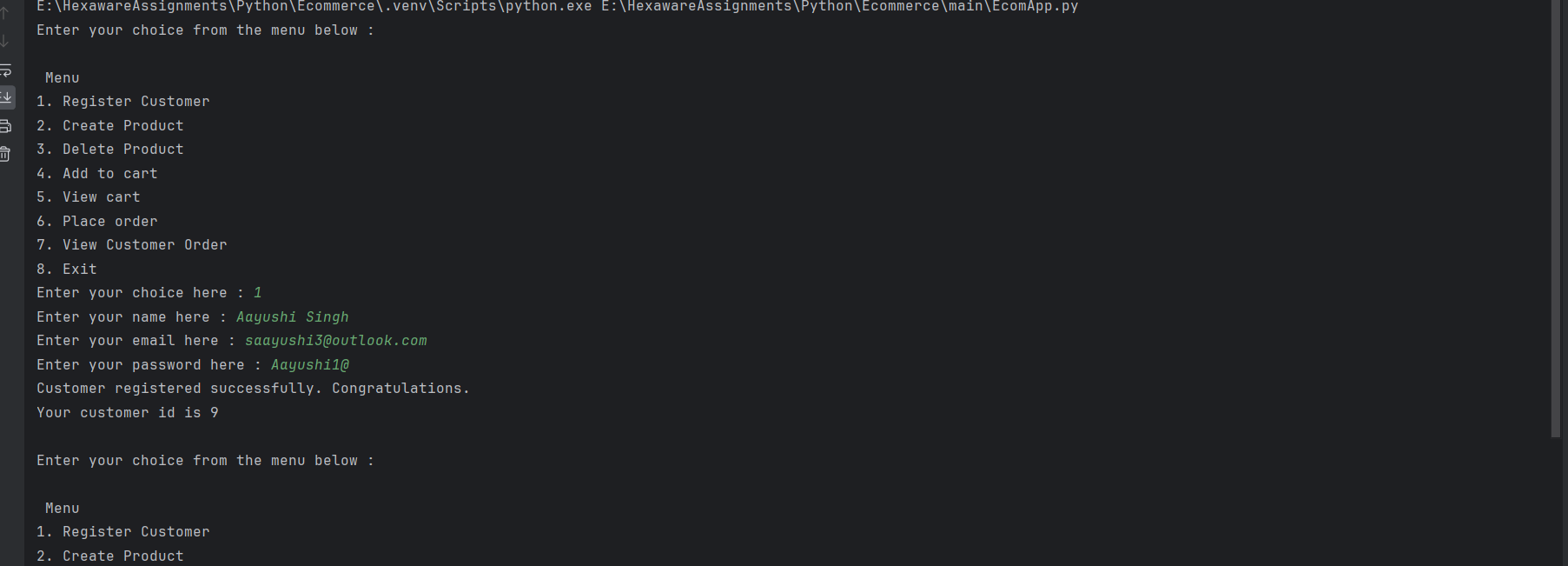
**Create class named EcomApp with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.**



1. **Register Customer.**

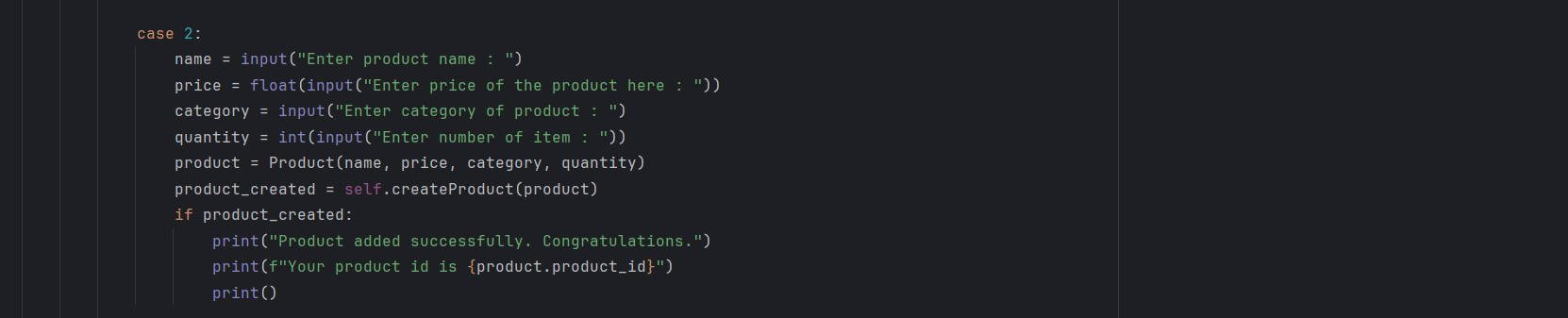


**Output:**



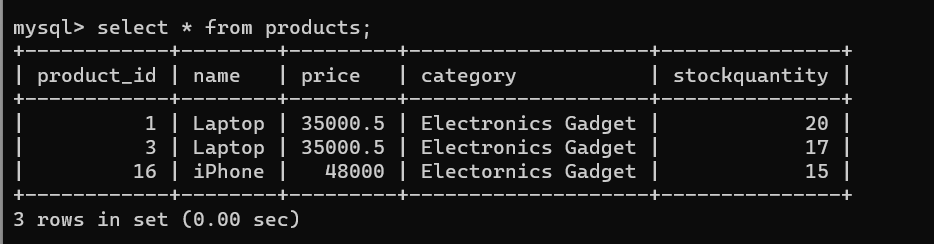


1. **Create Product.**

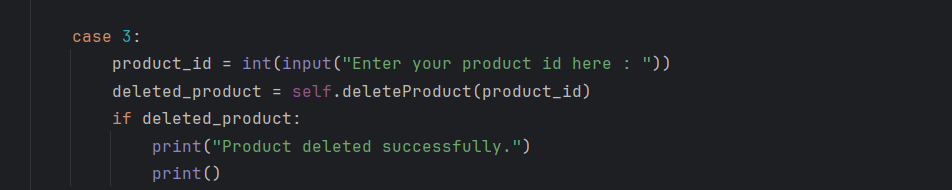


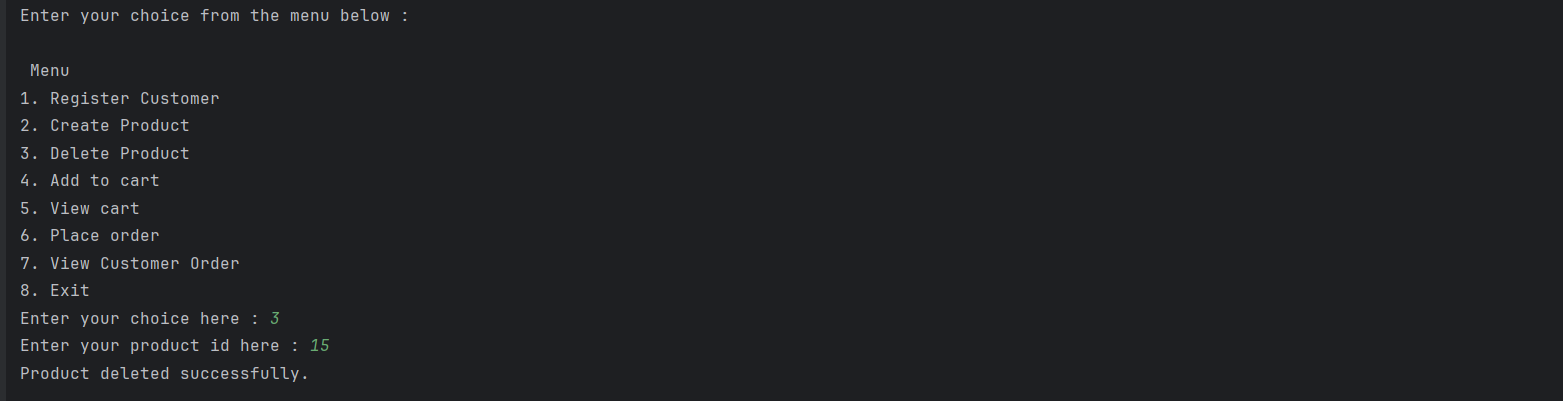
**Output:**



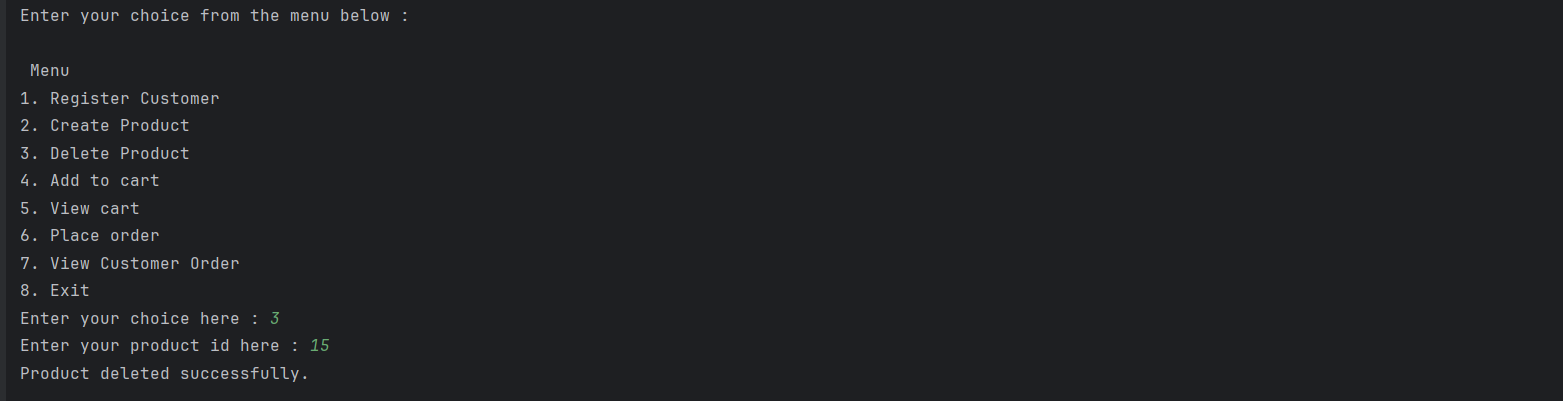


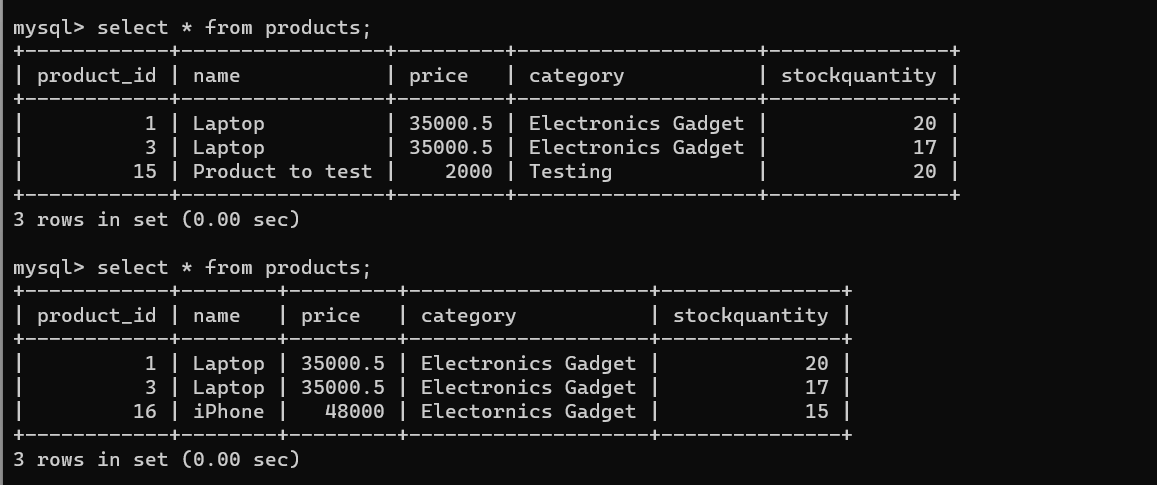
1. **Delete Product.**

****

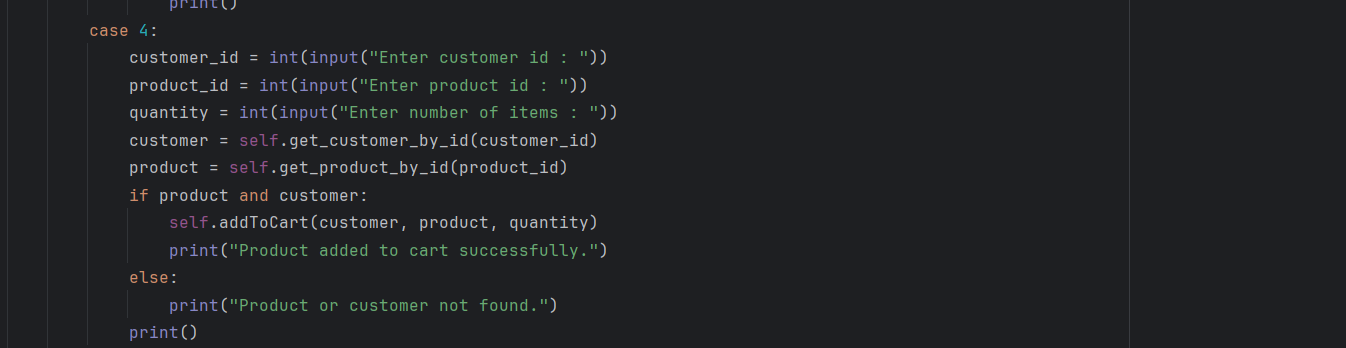


**Output:**



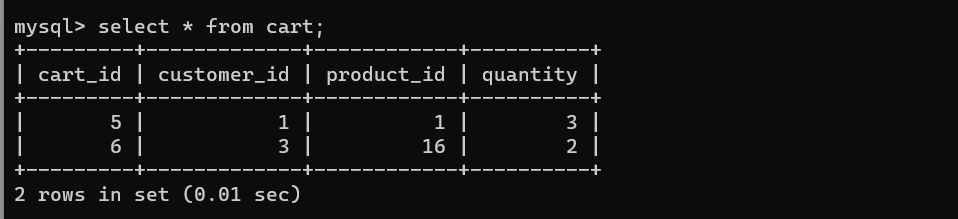


1. **Add to cart.**

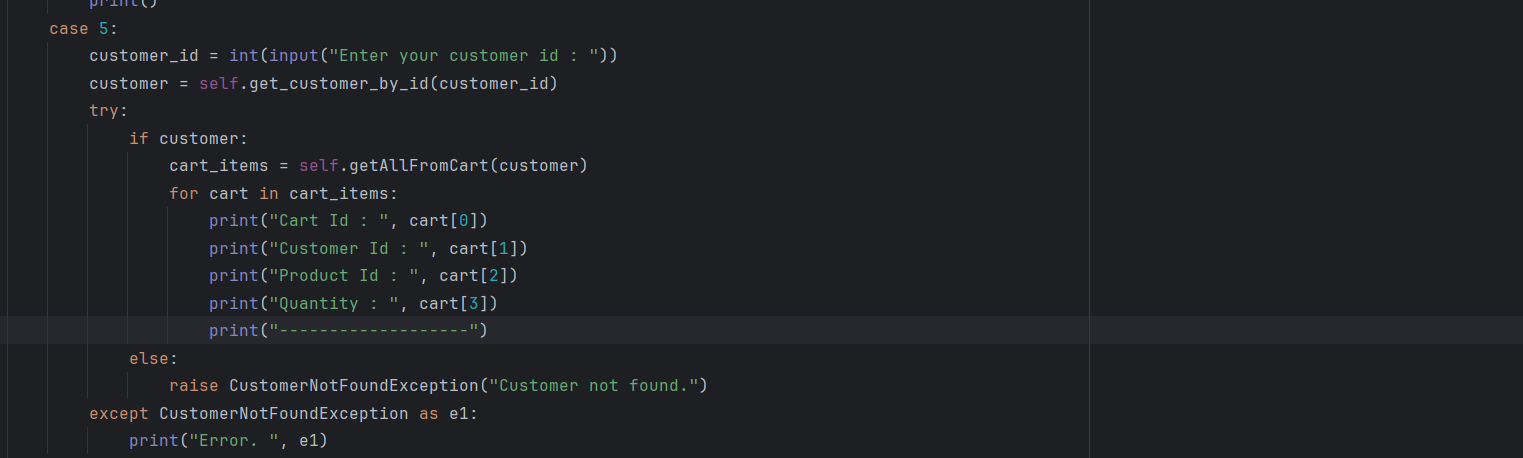
****

**Output :**

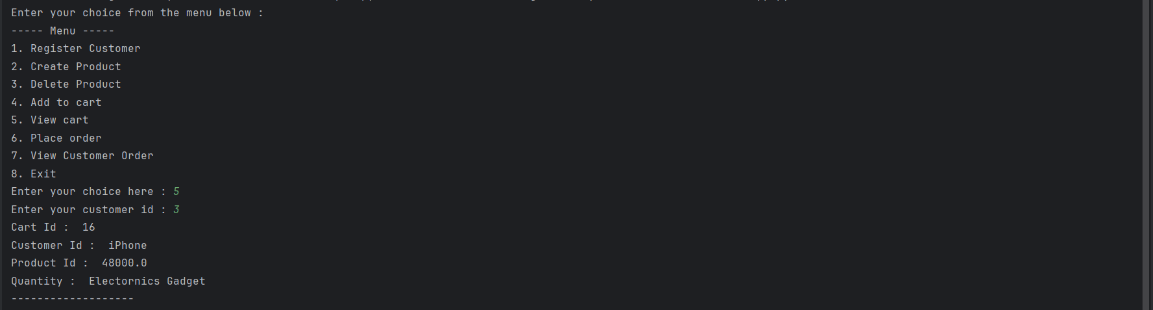


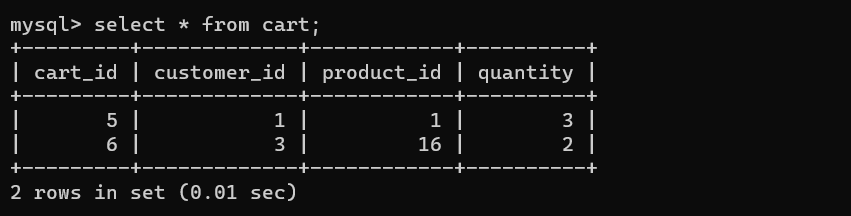


1. **View cart.**

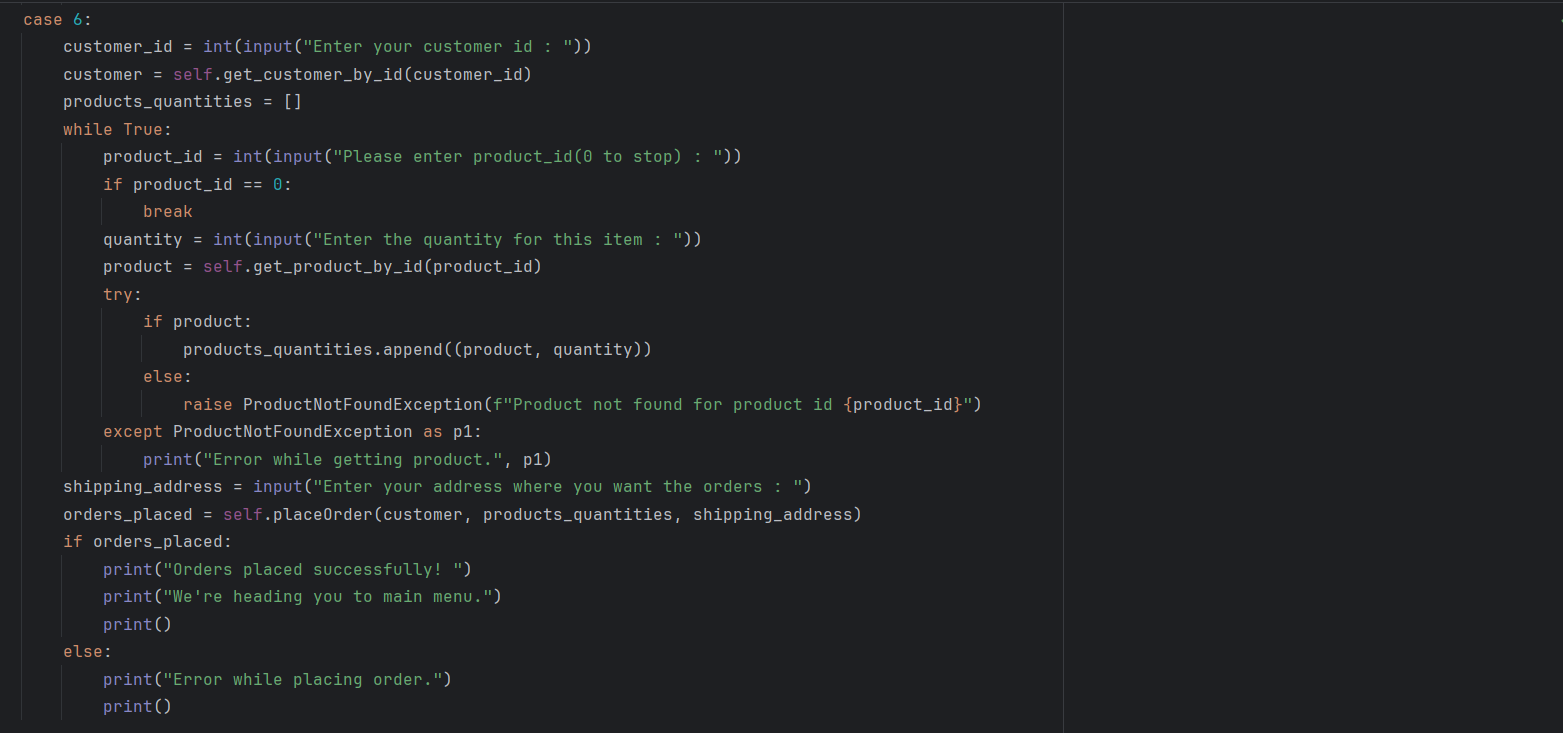
****

**Output :**

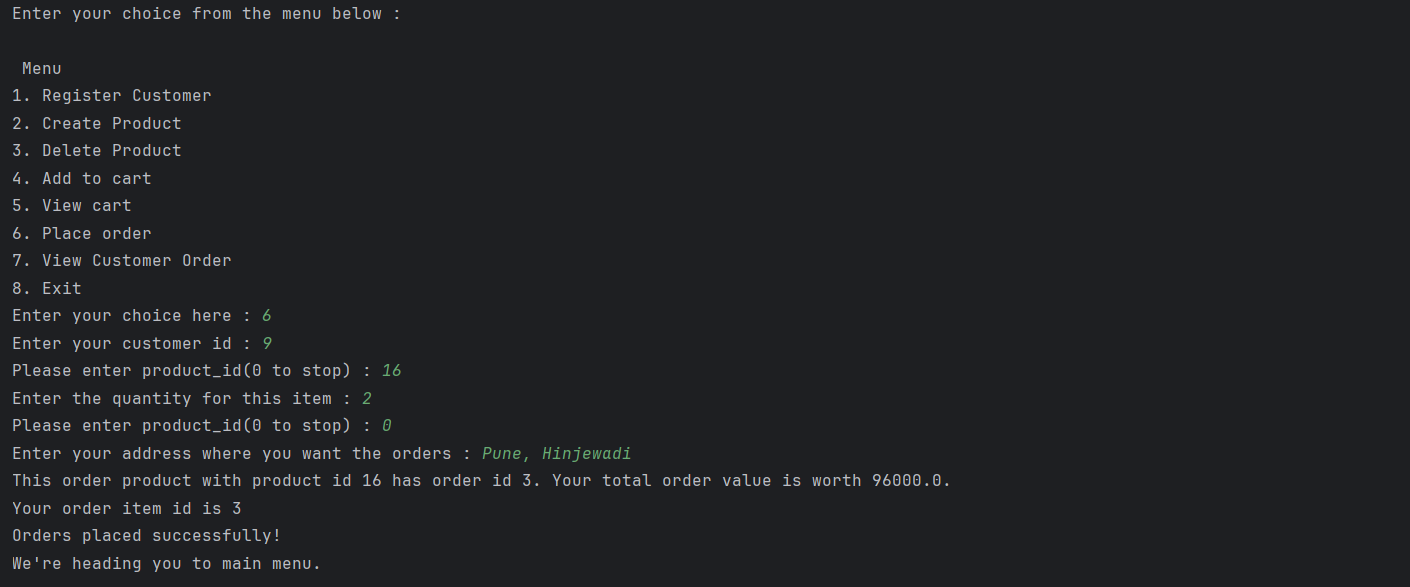
****

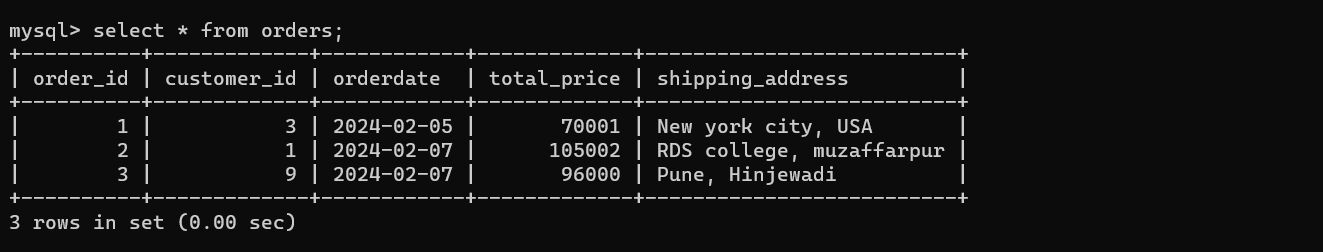
****

1. **Place order.**

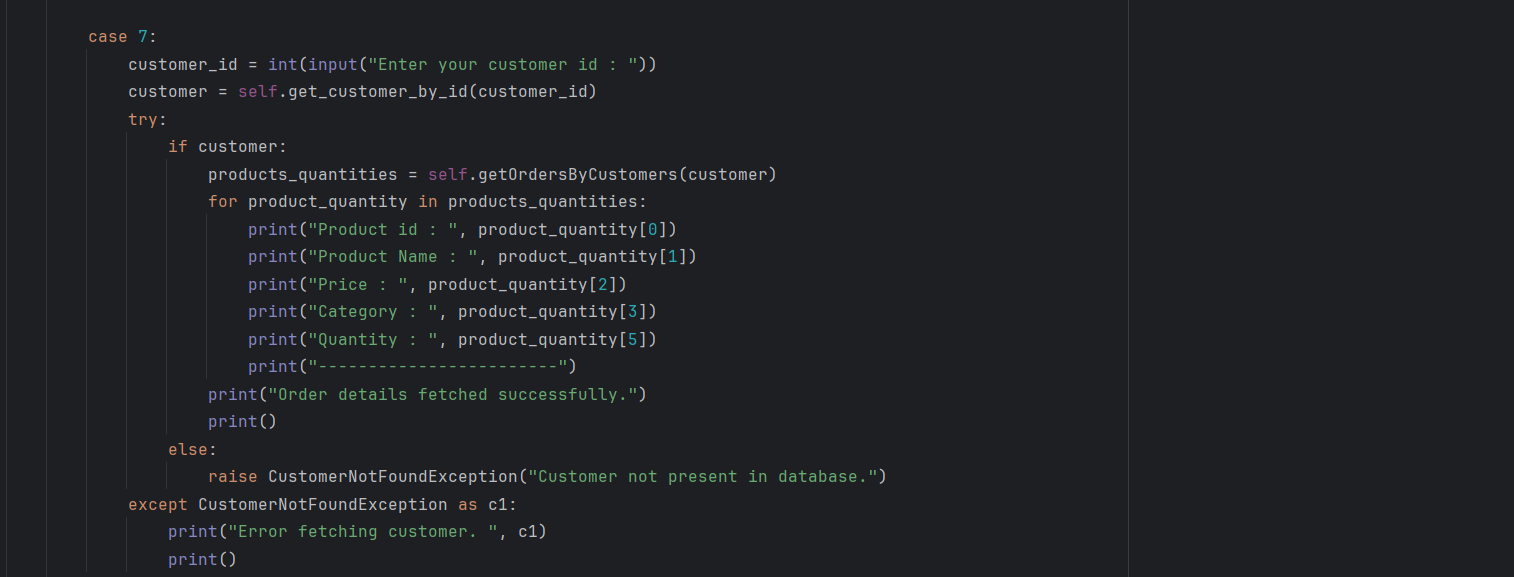
****

**Output:**

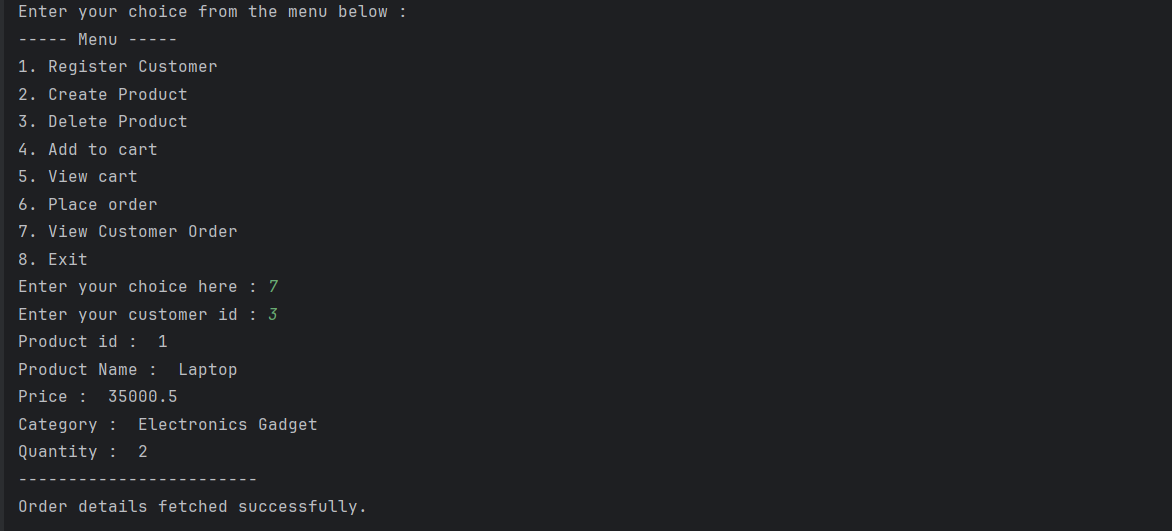
****

****

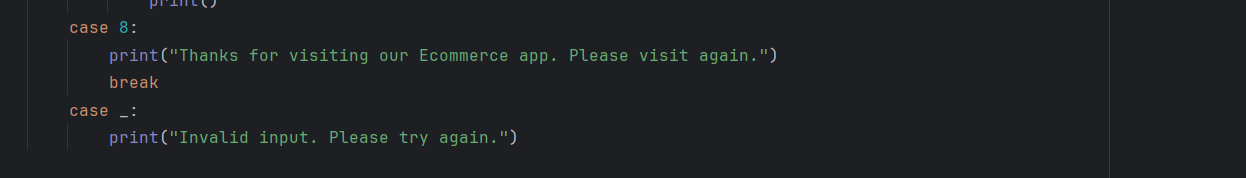
1. **View Customer Order**

****

**Output :**

****

1. **Exit**

****

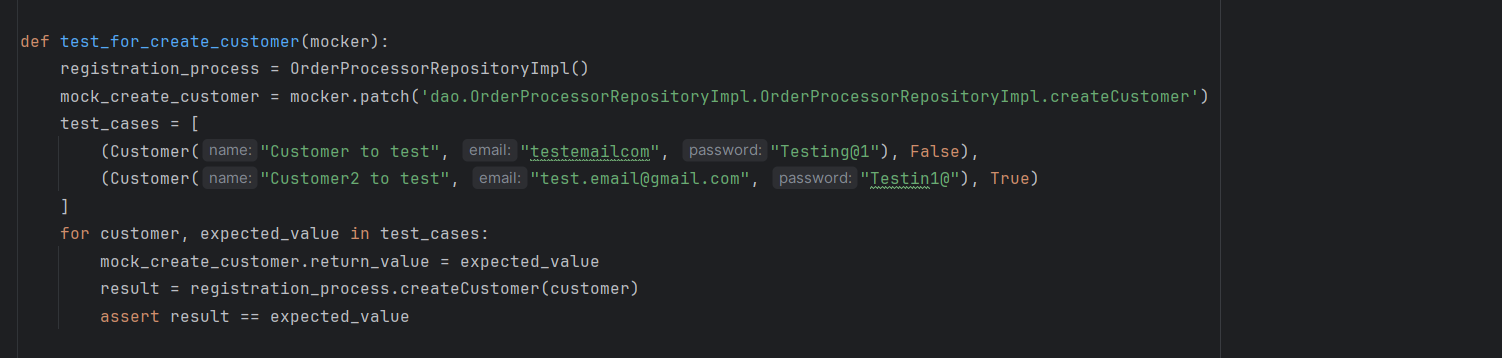
Unit Testing

Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

• Write test case to test Product created successfully or not.



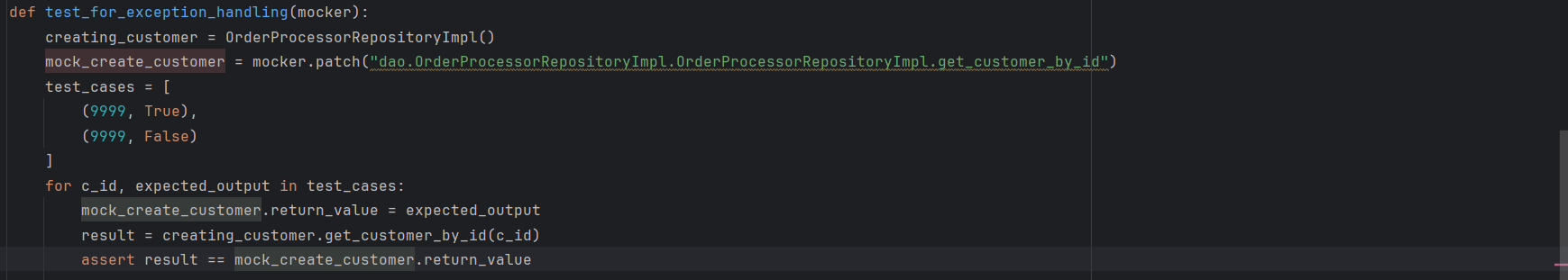
• Write test case to test product is added to cart successfully or not.



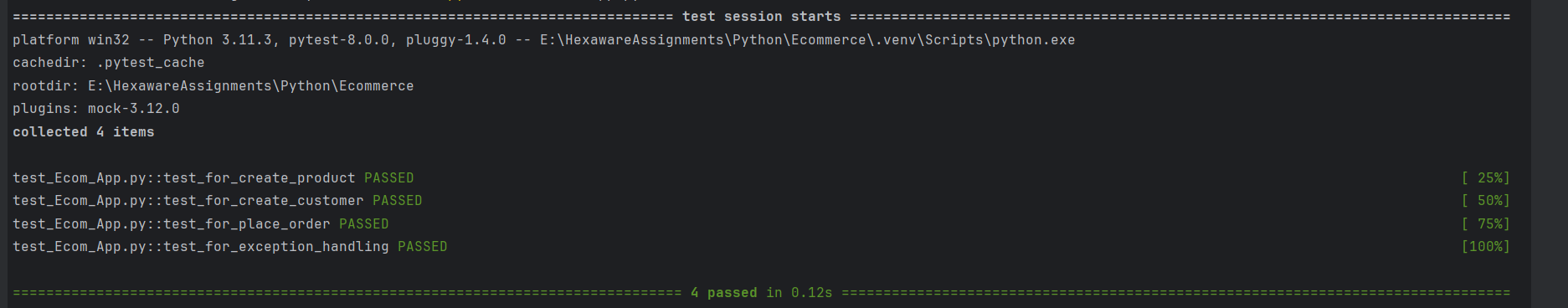
• Write test case to test product is ordered successfully or not.



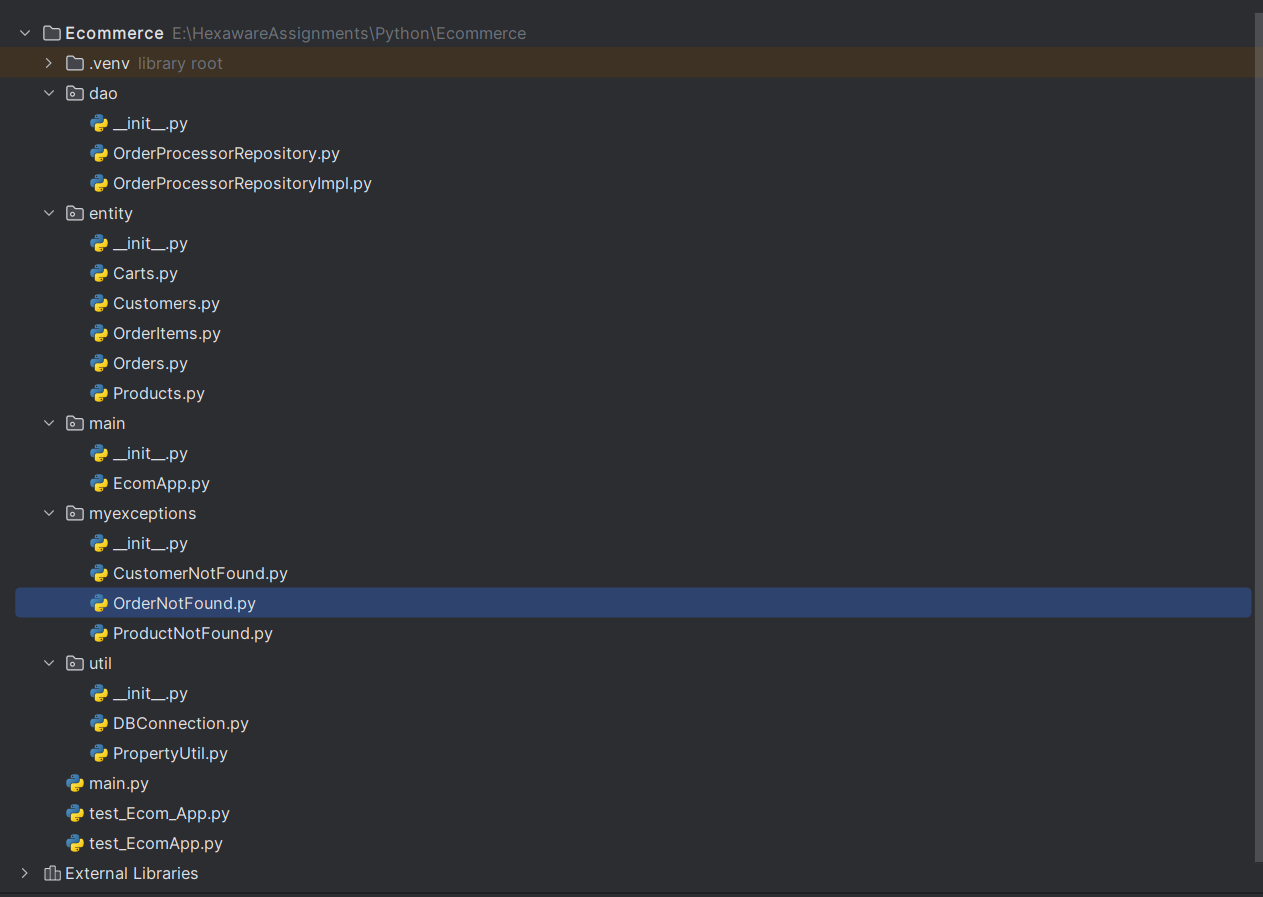
• Write test case to test exception is thrown correctly or not when customer id or product id not found in database.

****

**Output:**

****

**Package Management:**

****