

# Case Study on Ecommerce Application

BY SHREYASI REJA

The following Directory structure is to be followed in the application.

entity/model

Create entity classes in this package. All entity class should not have any business logic.

dao

Create Service Provider interface to showcase functionalities.

Create the implementation class for the above interface with db interaction.

exception

Create user defined exceptions in this package and handle exceptions whenever needed.

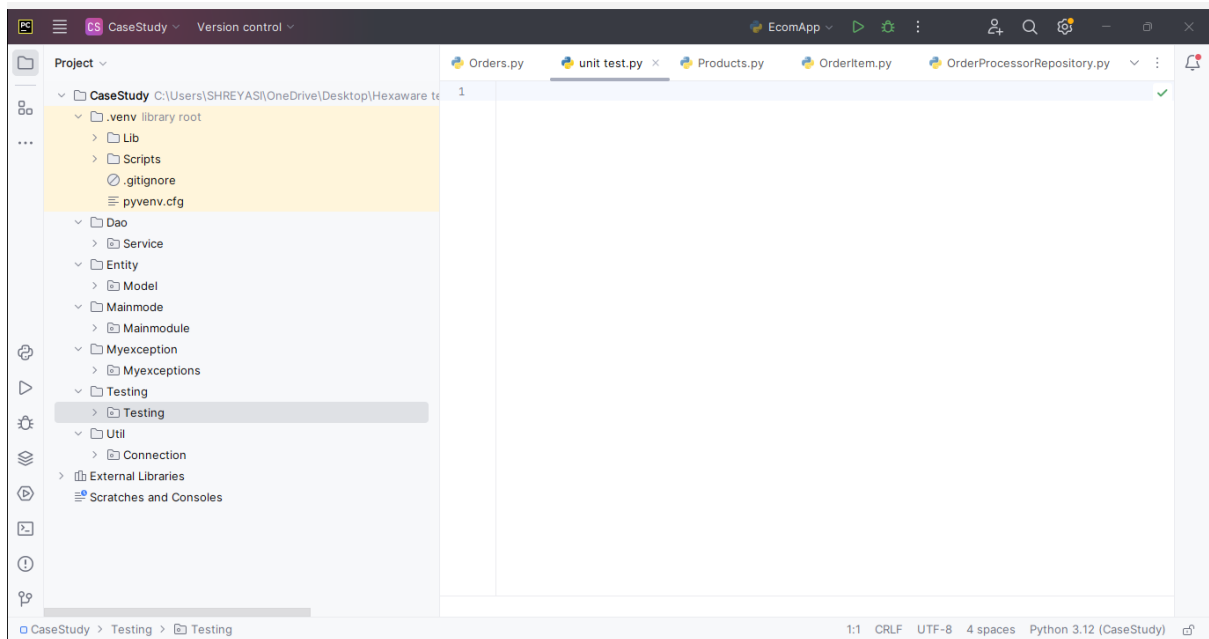
util

Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.

Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object(Use method defined in DBPropertyUtil class to get the connection String ).

main

Create a class MainModule and demonstrate the functionalities in a menu driven application.



Create following tables in SQL Schema with appropriate class and write the unit test case for the Ecommerce application.

Schema Design:

1. customers table:

customer\_id (Primary Key)

name

email

password

```
mysql> DESC CUSTOMERS;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| customer_id | int | NO | PRI | NULL | |
| name | varchar(255) | YES | | NULL | |
| email | varchar(255) | YES | | NULL | |
| password | varchar(255) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.06 sec)
```

2. products table:

?

product\_id (Primary Key)

name

price

description

## stockQuantity

```
mysql> DESC PRODUCTS;
```

Field	Type	Null	Key	Default	Extra
product_id	int	NO	PRI	NULL	
name	varchar(255)	YES		NULL	
price	decimal(10,2)	YES		NULL	
description	text	YES		NULL	
stockQuantity	int	YES		NULL	

5 rows in set (0.00 sec)

### 3. cart table:

cart\_id (Primary Key)

customer\_id (Foreign Key)

product\_id (Foreign Key)

quantity

```
mysql> DESC CART;
```

Field	Type	Null	Key	Default	Extra
cart_id	int	NO	PRI	NULL	
customer_id	int	YES	MUL	NULL	
product_id	int	YES	MUL	NULL	
quantity	int	YES		NULL	

4 rows in set (0.00 sec)

### 4. orders table:

order\_id (Primary Key)

customer\_id (Foreign Key)

order\_date

total\_price

shipping\_address

```
mysql> DESC ORDERS;
```

Field	Type	Null	Key	Default	Extra
order_id	int	NO	PRI	NULL	
customer_id	int	YES	MUL	NULL	
order_date	date	YES		NULL	
total_price	decimal(10,2)	YES		NULL	
shipping_address	varchar(255)	YES		NULL	

5 rows in set (0.00 sec)

```
mysql>
```

### 5. order\_items table (to store order details):

?

order\_item\_id (Primary Key)

order\_id (Foreign Key)

product\_id (Foreign Key)

quantity

```
mysql> DESC ORDER_ITEMS;
```

Field	Type	Null	Key	Default	Extra
order_item_id	int	NO	PRI	NULL	
order_id	int	YES	MUL	NULL	
product_id	int	YES	MUL	NULL	
quantity	int	YES		NULL	

4 rows in set (0.00 sec)

Create the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters, setters )

Customer:-

The screenshot shows a code editor with a project explorer on the left and a code editor on the right. The project explorer shows a directory structure for 'CaseStudy' with subdirectories like 'Lib', 'Scripts', 'pyenvv.cfg', 'Dao', 'Service', 'Entity', and 'Model'. The 'Model' directory contains files like '\_\_init\_\_.py', 'Cart.py', 'Customers.py', 'Orderitem.py', 'Orders.py', and 'Products.py'. The 'Customers.py' file is selected and its content is displayed in the code editor. The code defines a 'Customer' class with an '\_\_init\_\_' method, a 'customer\_id' property with a getter and setter, and a 'name' property with a getter and setter. The status bar at the bottom indicates the file is 'Customers.py' in the 'Entity > Model' directory, using 'Python 3.12 (CaseStudy)' with '15:1 CRLF UTF-8 4 spaces'.

```
class Customer:
    def __init__(self, customer_id=None, name=None, email=None, password=None):
        self._customer_id = customer_id
        self._name = name
        self._email = email
        self._password = password

    1 usage
    @property
    def customer_id(self):
        return self._customer_id

    @customer_id.setter
    def customer_id(self, value):
        self._customer_id = value

    1 usage
    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    1 usage
    @property
```

```
20 @name.setter
21 def name(self, value):
22     self.__name = value
23
24 1 usage
25 @property
26 def email(self):
27     return self.__email
28
29 @email.setter
30 def email(self, value):
31     self.__email = value
32
33 1 usage
34 @property
35 def password(self):
36     return self.__password
37
38 @password.setter
39 def password(self, value):
40     self.__password = value
```

Customer

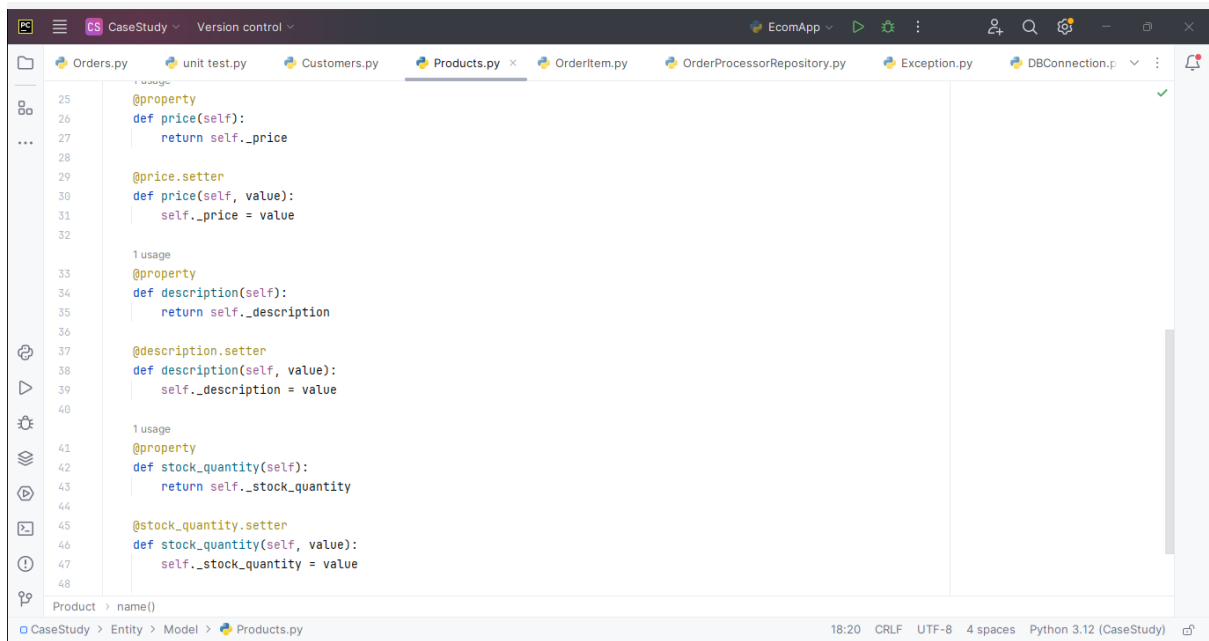
CaseStudy > Entity > Model > Customers.py 15:1 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

## Product:-

```
1 class Product:
2     def __init__(self, product_id=None, name=None, price=None, description=None, stock_quantity=None):
3         self.__product_id = product_id
4         self.__name = name
5         self.__price = price
6         self.__description = description
7         self.__stock_quantity = stock_quantity
8
9 1 usage
10 @property
11 def product_id(self):
12     return self.__product_id
13
14 @product_id.setter
15 def product_id(self, value):
16     self.__product_id = value
17
18 1 usage
19 @property
20 def name(self):
21     return self.__name
22
23 @name.setter
24 def name(self, value):
25     self.__name = value
```

Product > name()

CaseStudy > Entity > Model > Products.py 18:20 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)



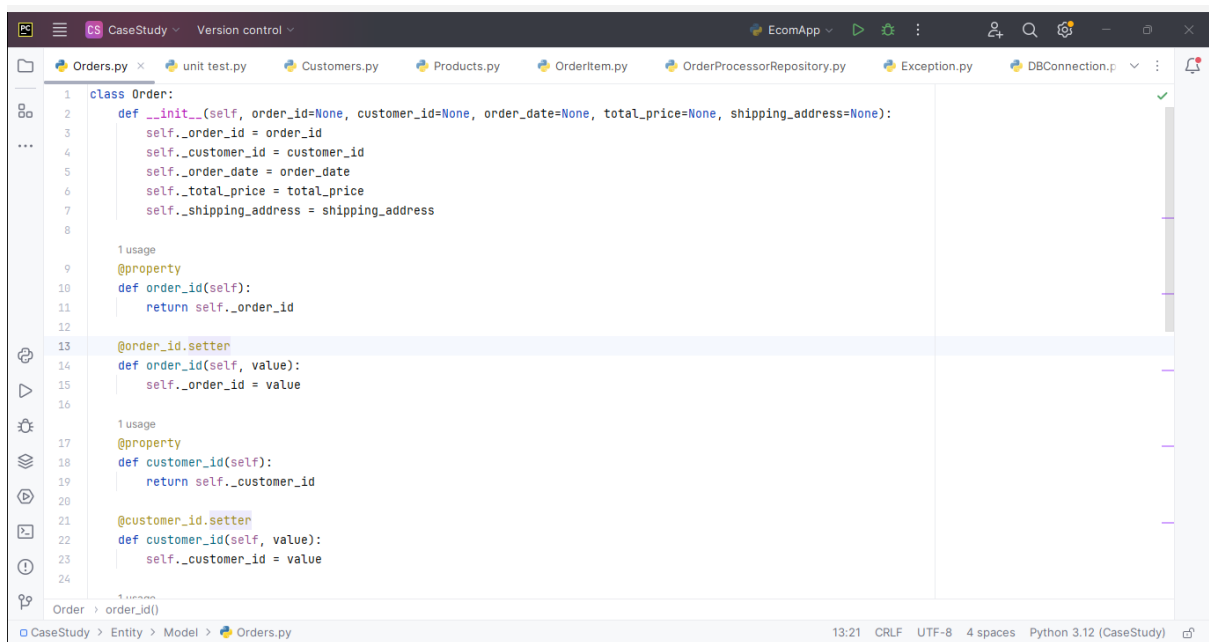
```
25 @property
26 def price(self):
27     return self._price
28
29 @price.setter
30 def price(self, value):
31     self._price = value
32
33 1 usage
34 @property
35 def description(self):
36     return self._description
37
38 @description.setter
39 def description(self, value):
40     self._description = value
41
42 1 usage
43 @property
44 def stock_quantity(self):
45     return self._stock_quantity
46
47 @stock_quantity.setter
48 def stock_quantity(self, value):
49     self._stock_quantity = value
```

Product > name()

CaseStudy > Entity > Model > Products.py

18:20 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

## Orders:-

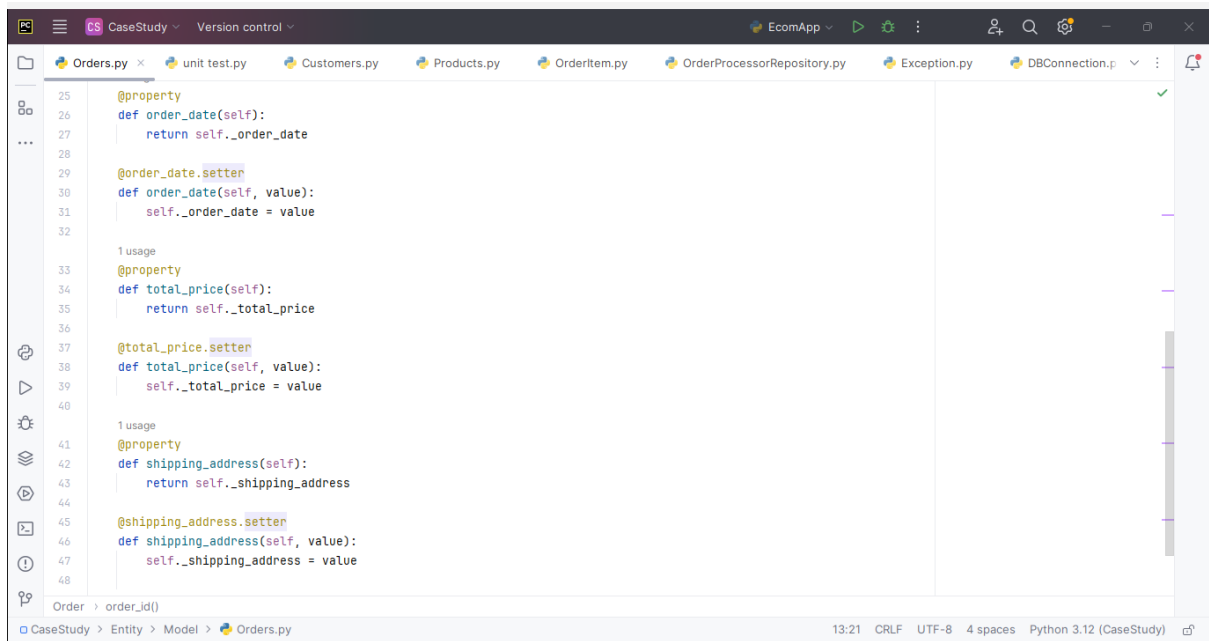


```
1 class Order:
2     def __init__(self, order_id=None, customer_id=None, order_date=None, total_price=None, shipping_address=None):
3         self._order_id = order_id
4         self._customer_id = customer_id
5         self._order_date = order_date
6         self._total_price = total_price
7         self._shipping_address = shipping_address
8
9 1 usage
10 @property
11 def order_id(self):
12     return self._order_id
13
14 @order_id.setter
15 def order_id(self, value):
16     self._order_id = value
17
18 1 usage
19 @property
20 def customer_id(self):
21     return self._customer_id
22
23 @customer_id.setter
24 def customer_id(self, value):
25     self._customer_id = value
```

Order > order\_id()

CaseStudy > Entity > Model > Orders.py

13:21 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

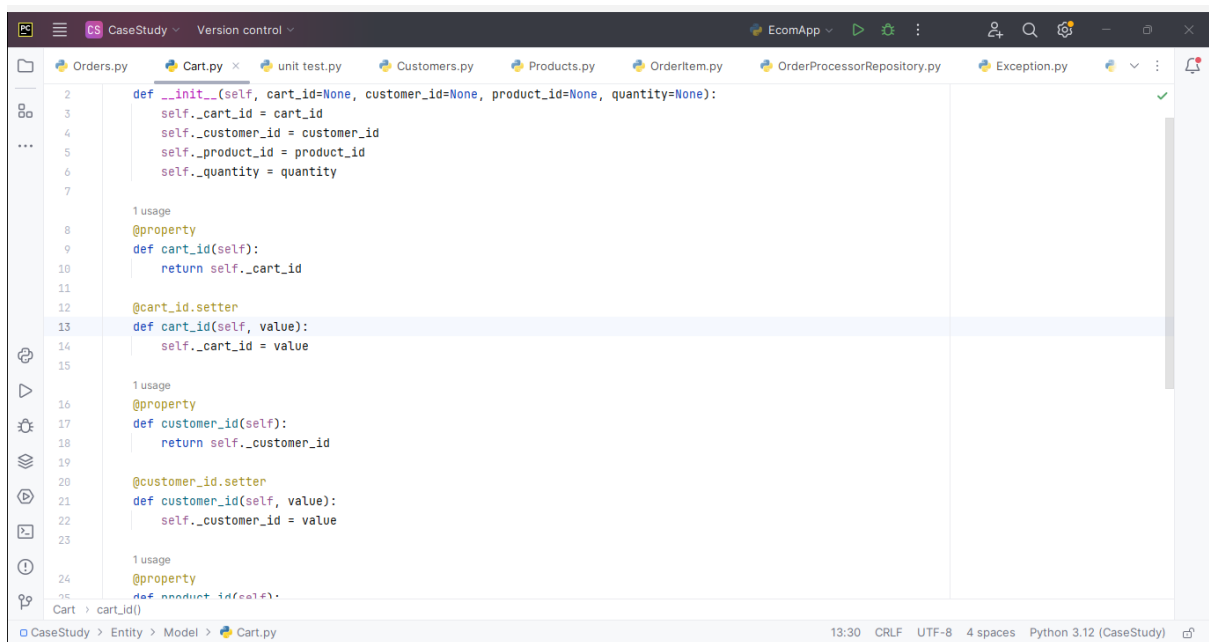


```
25 @property
26 def order_date(self):
27     return self._order_date
28
29 @order_date.setter
30 def order_date(self, value):
31     self._order_date = value
32
33 1 usage
34 @property
35 def total_price(self):
36     return self._total_price
37
38 @total_price.setter
39 def total_price(self, value):
40     self._total_price = value
41
42 1 usage
43 @property
44 def shipping_address(self):
45     return self._shipping_address
46
47 @shipping_address.setter
48 def shipping_address(self, value):
49     self._shipping_address = value
```

Order > order\_id()

CaseStudy > Entity > Model > Orders.py 13:21 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

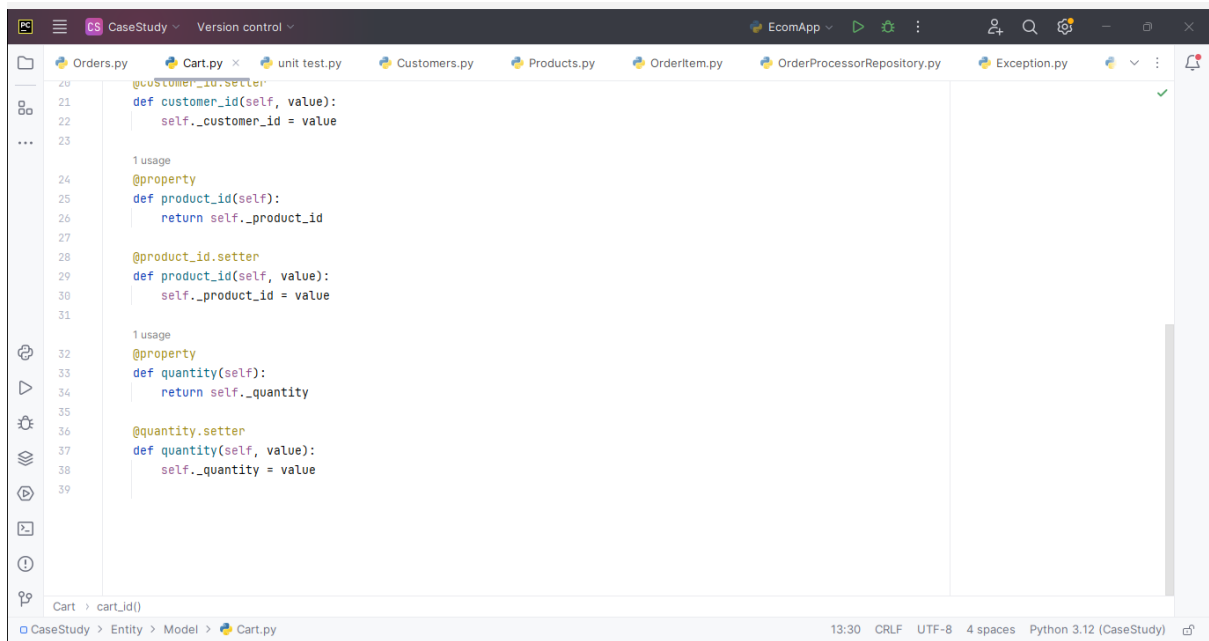
## Cart:-



```
2 def __init__(self, cart_id=None, customer_id=None, product_id=None, quantity=None):
3     self._cart_id = cart_id
4     self._customer_id = customer_id
5     self._product_id = product_id
6     self._quantity = quantity
7
8 1 usage
9 @property
10 def cart_id(self):
11     return self._cart_id
12
13 @cart_id.setter
14 def cart_id(self, value):
15     self._cart_id = value
16
17 1 usage
18 @property
19 def customer_id(self):
20     return self._customer_id
21
22 @customer_id.setter
23 def customer_id(self, value):
24     self._customer_id = value
25
26 1 usage
27 @property
28 def product_id(self):
29     return self._product_id
```

Cart > cart\_id()

CaseStudy > Entity > Model > Cart.py 13:30 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

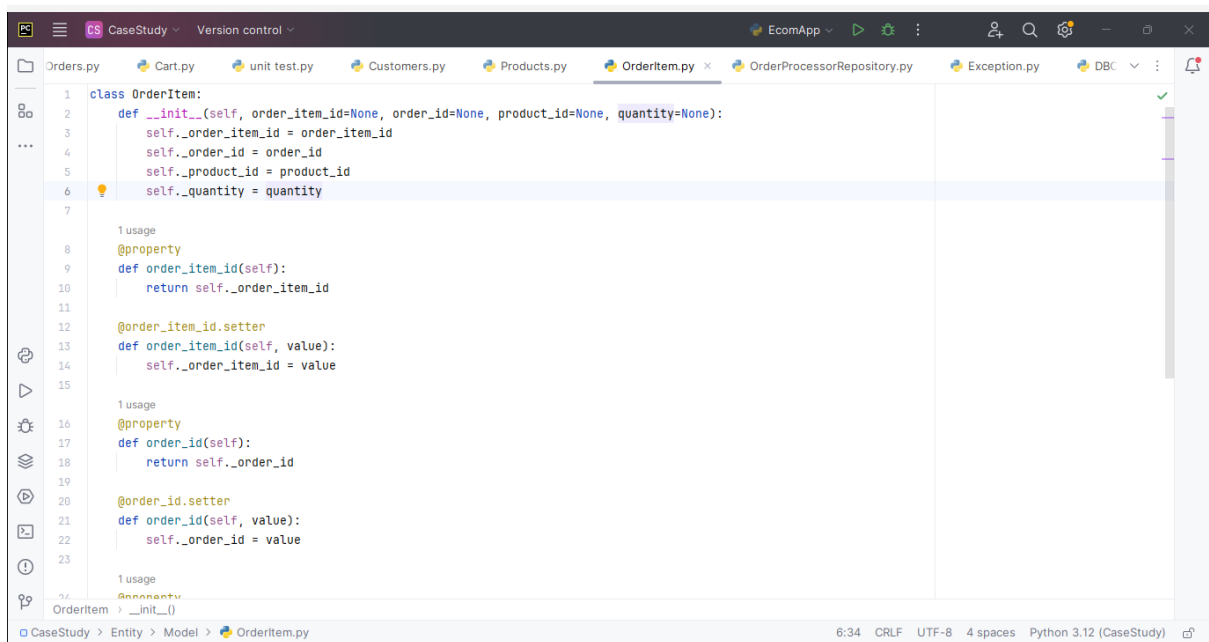


```
20 @customer_id.setter
21 def customer_id(self, value):
22     self._customer_id = value
23
24 1 usage
25 @property
26 def product_id(self):
27     return self._product_id
28
29 @product_id.setter
30 def product_id(self, value):
31     self._product_id = value
32
33 1 usage
34 @property
35 def quantity(self):
36     return self._quantity
37
38 @quantity.setter
39 def quantity(self, value):
40     self._quantity = value
```

CaseStudy > Entity > Model > Cart.py

13:30 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

## Orderitem:-

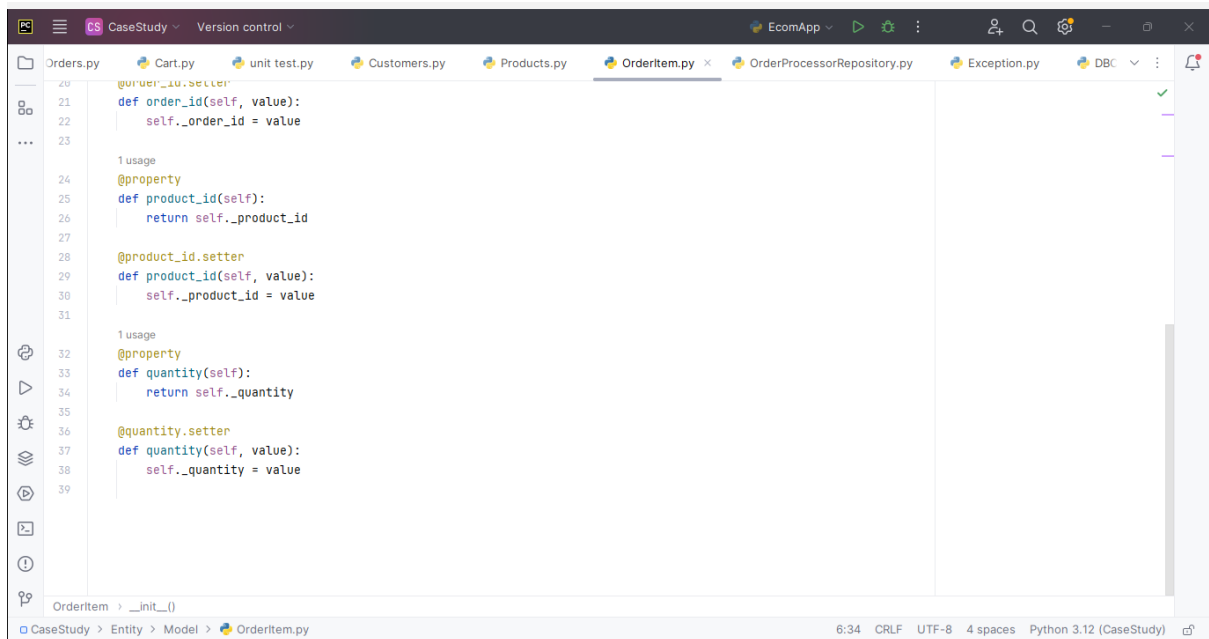


```
1 class OrderItem:
2     def __init__(self, order_item_id=None, order_id=None, product_id=None, quantity=None):
3         self._order_item_id = order_item_id
4         self._order_id = order_id
5         self._product_id = product_id
6         self._quantity = quantity
7
8 1 usage
9 @property
10 def order_item_id(self):
11     return self._order_item_id
12
13 @order_item_id.setter
14 def order_item_id(self, value):
15     self._order_item_id = value
16
17 1 usage
18 @property
19 def order_id(self):
20     return self._order_id
21
22 @order_id.setter
23 def order_id(self, value):
24     self._order_id = value
25
26 1 usage
27 @property
28 def product_id(self):
29     return self._product_id
30
31 @product_id.setter
32 def product_id(self, value):
33     self._product_id = value
34
35 1 usage
36 @property
37 def quantity(self):
38     return self._quantity
39
40 @quantity.setter
41 def quantity(self, value):
42     self._quantity = value
```

CaseStudy > Entity > Model > OrderItem.py

6:34 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)





```
20 @order_id.setter
21 def order_id(self, value):
22     self._order_id = value
23
24 1 usage
25 @property
26 def product_id(self):
27     return self._product_id
28
29 @product_id.setter
30 def product_id(self, value):
31     self._product_id = value
32
33 1 usage
34 @property
35 def quantity(self):
36     return self._quantity
37
38 @quantity.setter
39 def quantity(self, value):
40     self._quantity = value
```

## 6. 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<Product,quantity>>, string shippingAddress): should update order table and orderItems table.

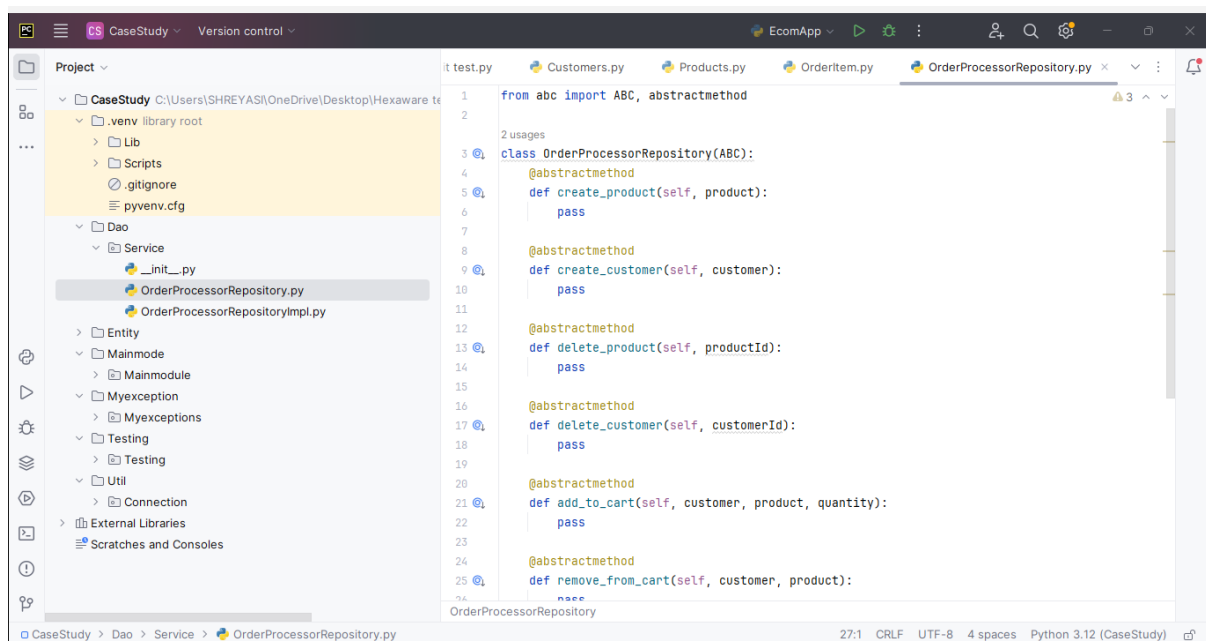
1. parameter: Customer customer, list of product and quantity

2. return type: boolean

9. getOrdersByCustomer()

1. parameter: customerid

2. return type: list of product and quantity



```
18 pass
19
20 @abstractmethod
21 def add_to_cart(self, customer, product, quantity):
22     pass
23
24 @abstractmethod
25 def remove_from_cart(self, customer, product):
26     pass
27
28 @abstractmethod
29 def get_all_from_cart(self, customer):
30     pass
31
32 @abstractmethod
33 def place_order(self, customer, products_quantity, shipping_address):
34     pass
35
36 @abstractmethod
37 def get_orders_by_customer(self, customer_id):
38     pass
39
```

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

```
1 from Dao.Service.OrderProcessorRepository import OrderProcessorRepository
2 from Myexception.Myexceptions.Exception import CustomerNotFoundException, ProductNotFoundException
3
4 2 usages
5 class OrderProcessorRepositoryImpl(OrderProcessorRepository):
6     def __init__(self):
7         self.db_util = {}
8         self.customers_database = {}
9         self.products_database = {}
10        self.cart_database = {}
11        self.orders_database = {}
12        self.order_items_database = {}
13
14    def create_product(self, product):
15        product_id = len(self.products_database) + 1
16        self.products_database[product_id] = product
17        return product_id
18
19    def create_customer(self, customer):
20        customer_id = len(self.customers_database) + 1
21        self.customers_database[customer_id] = customer
22        return customer_id
23
24    def delete_product(self, product_id):
25        if product_id not in self.products_database:
26            raise ProductNotFoundException("Product not found in the database.")
27        del self.products_database[product_id]
28
29    def remove_from_cart(self):
```

```
28 def delete_customer(self, customer_id):
29     if customer_id not in self.customers_database:
30         raise CustomerNotFoundException("Customer not found in the database.")
31     del self.customers_database[customer_id]
32
33 def add_to_cart(self, customer, product, quantity):
34     if product.product_id not in self.products_database:
35         raise ProductNotFoundException("Product not found in the database.")
36
37     if customer.customer_id not in self.cart_database:
38         self.cart_database[customer.customer_id] = {}
39
40     if product.product_id in self.cart_database[customer.customer_id]:
41         self.cart_database[customer.customer_id][product.product_id] += quantity
42     else:
43         self.cart_database[customer.customer_id][product.product_id] = quantity
44
45 def remove_from_cart(self, customer, product):
46     if customer.customer_id not in self.cart_database or product.product_id not in self.cart_database[
47         customer.customer_id]:
48         raise ProductNotFoundException("Product not found in the cart.")
49
50     del self.cart_database[customer.customer_id][product.product_id]
51
52 def get_all_from_cart(self, customer):
53     return self.cart_database.get(customer.customer_id, {})
```

OrderProcessorRepositoryImpl > remove\_from\_cart()

CaseStudy > Dao > Service > OrderProcessorRepositoryImpl.py 50:71 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```
43 self.cart_database[customer.customer_id][product.product_id] = quantity
44
45 def remove_from_cart(self, customer, product):
46     if customer.customer_id not in self.cart_database or product.product_id not in self.cart_database[
47         customer.customer_id]:
48         raise ProductNotFoundException("Product not found in the cart.")
49
50     del self.cart_database[customer.customer_id][product.product_id]
51
52 def get_all_from_cart(self, customer):
53     return self.cart_database.get(customer.customer_id, {})
54
55 def place_order(self, customer, products_quantities):
56     order_id = len(self.orders_database) + 1
57     self.orders_database[order_id] = {'customer_id': customer.customer_id,
58                                     'products_quantities': products_quantities}
59     return order_id
60
61 def get_orders_by_customer(self, customer_id):
62     user_orders = [order for order in self.orders_database.values() if order['customer_id'] == customer_id]
63     return user_orders
64
```

OrderProcessorRepositoryImpl > remove\_from\_cart()

CaseStudy > Dao > Service > OrderProcessorRepositoryImpl.py 50:71 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

Connect your application to the SQL database:

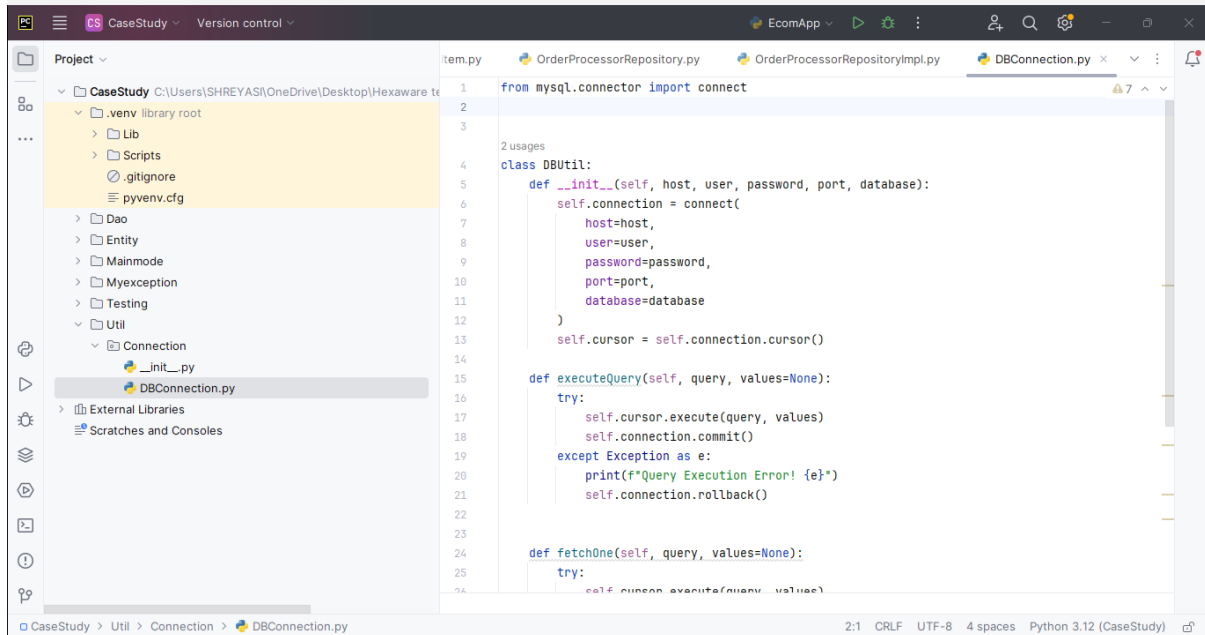
8. Write code to establish a connection to your SQL database.

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.

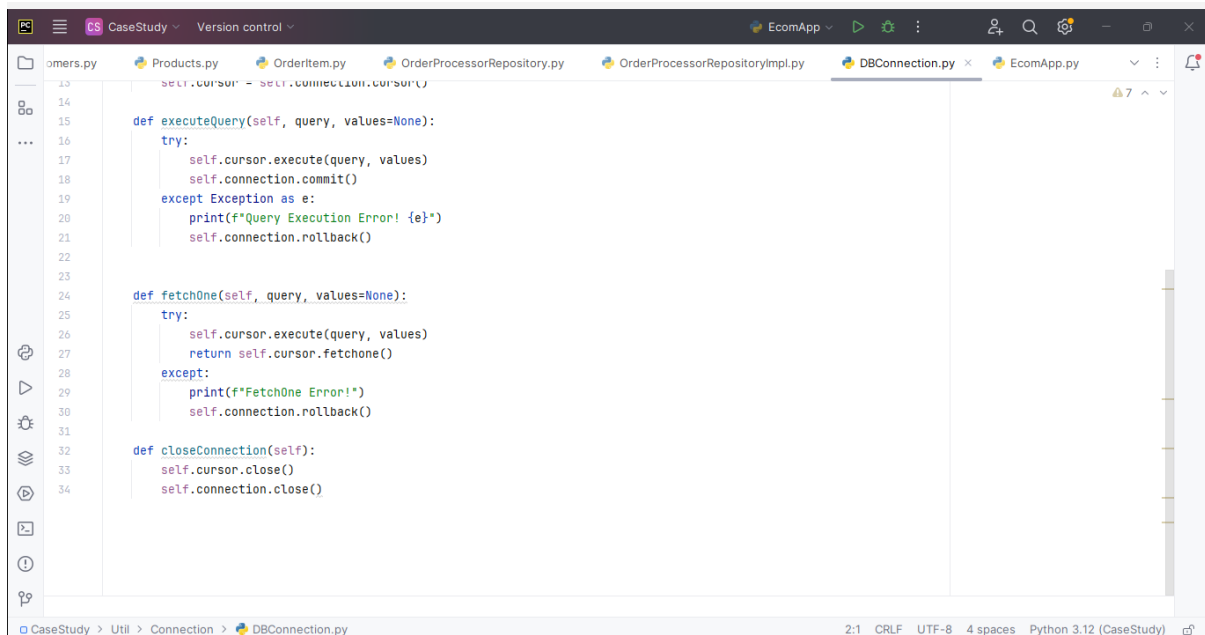
Connection properties supplied in the connection string should be read from a property file.

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.



```
1 from mysql.connector import connect
2
3
4 2 usages
5 class DBUtil:
6     def __init__(self, host, user, password, port, database):
7         self.connection = connect(
8             host=host,
9             user=user,
10            password=password,
11            port=port,
12            database=database
13        )
14        self.cursor = self.connection.cursor()
15
16     def executeQuery(self, query, values=None):
17         try:
18             self.cursor.execute(query, values)
19             self.connection.commit()
20         except Exception as e:
21             print(f"Query Execution Error! {e}")
22             self.connection.rollback()
23
24     def fetchOne(self, query, values=None):
25         try:
26             self.cursor.execute(query, values)
```



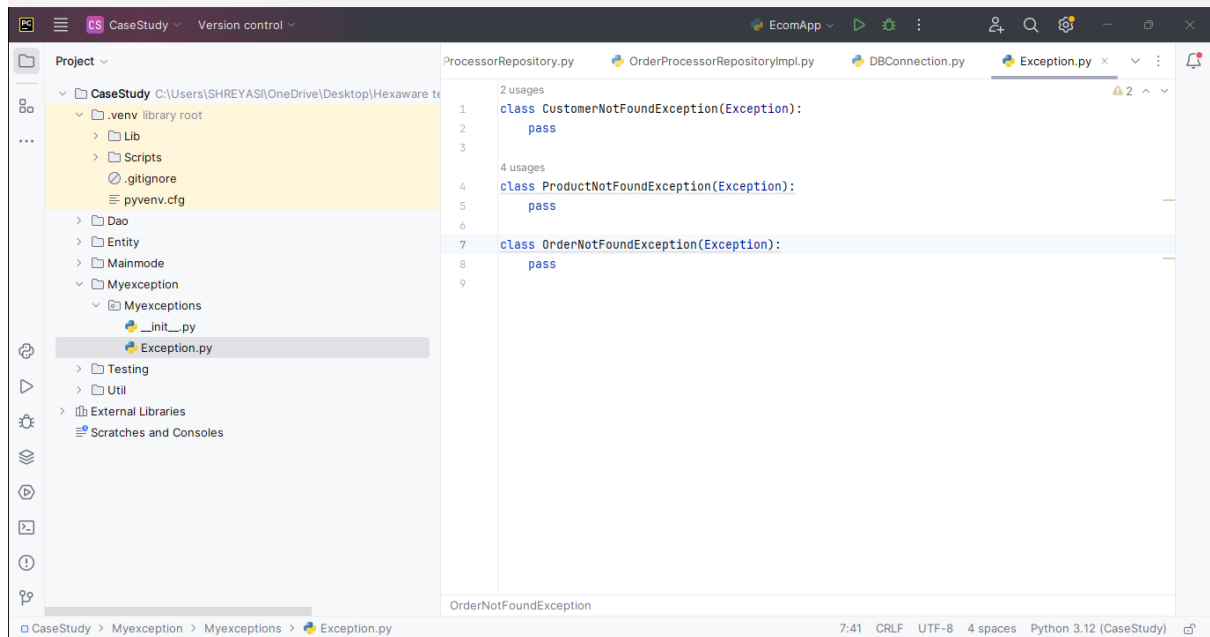
```
23 self.cursor = self.connection.cursor()
24
25 def executeQuery(self, query, values=None):
26     try:
27         self.cursor.execute(query, values)
28         self.connection.commit()
29     except Exception as e:
30         print(f"Query Execution Error! {e}")
31         self.connection.rollback()
32
33 def fetchOne(self, query, values=None):
34     try:
35         self.cursor.execute(query, values)
36         return self.cursor.fetchone()
37     except:
38         print(f"FetchOne Error!")
39         self.connection.rollback()
40
41 def closeConnection(self):
42     self.cursor.close()
43     self.connection.close()
```

9. 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



10. 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.
2. Create Product.
3. Delete Product.
4. Add to cart.
5. View cart.
6. Place order.
7. View Customer Order

```
from Dao.Service.OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl
from Util.Connection.DBConnection import DBUtil

1 usage
class EcomApp:
    def __init__(self):
        self.order_processor = OrderProcessorRepositoryImpl()
        self.db_util = DBUtil(host='localhost', user='root', password='Root', port='3306', da

1 usage
    def register_customer(self):
        try:
            connection = self.db_util.connection
            cursor = connection.cursor()

            customer_id = int(input("Enter customer ID: "))

            name = input("Enter customer name: ")
            email = input("Enter customer email: ")
            password = input("Enter customer password: ")

            query = "INSERT INTO customers (customer_id, name, email, password) VALUES (%s, %s, %s, %s)"
            cursor.execute(query, (customer_id, name, email, password))

            connection.commit()
            cursor.close()
            connection.close()
```

```
cursor.close()
connection.close()

print("Customer registered successfully!")
except Exception as e:
    print(f"Error registering customer: {e}")

1 usage
def create_product(self):
    try:
        connection = self.db_util.connection
        cursor = connection.cursor()

        name = input("Enter product name: ")
        price = float(input("Enter product price: "))
        description = input("Enter product description: ")
        stock_quantity = int(input("Enter product stock quantity: "))

        query = "INSERT INTO products (name, price, description, stock_quantity) VALUES (%s, %s, %s, %s)"
        cursor.execute(query, (name, price, description, stock_quantity))

        connection.commit()
        cursor.close()
        connection.close()

        print("Product created successfully!")
    except Exception as e:
```

This screenshot shows a code editor with the file `EcomApp.py` open. The `delete_product` method is defined, which takes `self` as an argument. It uses a try-except block to handle database operations. Inside the try block, it establishes a database connection, prompts the user for a product ID to delete, constructs a SQL query to delete the product, executes the query, commits the transaction, and closes the connection. It then prints a success message. The except block catches any exceptions and prints an error message. A usage example is provided below the method definition.

```
48         print("Product created successfully!")
49     except Exception as e:
50         print(f"Error creating product: {e}")
51
52     1 usage
53     def delete_product(self):
54         try:
55             connection = self.db_util.connection
56             cursor = connection.cursor()
57
58             product_id = int(input("Enter product ID to delete: "))
59
60             query = "DELETE FROM products WHERE product_id = %s"
61             cursor.execute(query, (product_id,))
62
63             connection.commit()
64             cursor.close()
65             connection.close()
66
67             print("Product deleted successfully!")
68         except Exception as e:
69             print(f"Error deleting product: {e}")
70
71     1 usage
72     def add_to_cart(self):
73         try:
```

The status bar at the bottom indicates the file is `EcomApp.py`, the cursor is at line 70, column 128:40, and the encoding is CRLF, UTF-8, with 4 spaces and Python 3.12 (CaseStudy).

This screenshot shows the same code editor with `EcomApp.py` open, displaying the `add_to_cart` and `view_cart` methods. The `add_to_cart` method uses a try-except block to handle database operations. It prompts the user for customer ID, product ID, and quantity, constructs an SQL query to insert the product into the cart, executes the query, commits the transaction, and closes the connection. It then prints a success message. The except block catches any exceptions and prints an error message. The `view_cart` method is also shown, which uses a try block to establish a database connection and retrieve the cart contents. A usage example is provided below the method definition.

```
70     def add_to_cart(self):
71         try:
72             connection = self.db_util.connection
73             cursor = connection.cursor()
74
75             customer_id = int(input("Enter customer ID: "))
76             product_id = int(input("Enter product ID to add to cart: "))
77             quantity = int(input("Enter quantity: "))
78
79             query = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s, %s)"
80             cursor.execute(query, (customer_id, product_id, quantity))
81
82             connection.commit()
83             cursor.close()
84             connection.close()
85
86             print("Product added to cart successfully!")
87         except Exception as e:
88             print(f"Error adding product to cart: {e}")
89
90     1 usage
91     def view_cart(self):
92         try:
93             connection = self.db_util.connection
94             cursor = connection.cursor()
95
96             cursor.execute("SELECT * FROM cart")
97             cart_items = cursor.fetchall()
98
99             if not cart_items:
100                 print("Cart is empty")
101             else:
102                 for item in cart_items:
103                     print(f"Customer ID: {item[0]}, Product ID: {item[1]}, Quantity: {item[2]}")
104
105             cursor.close()
106             connection.close()
107
108             print("Cart contents:")
109
110             1 usage
```

The status bar at the bottom indicates the file is `EcomApp.py`, the cursor is at line 90, column 128:40, and the encoding is CRLF, UTF-8, with 4 spaces and Python 3.12 (CaseStudy).



```
CaseStudy Version control EcomApp
ducts.py OrderItem.py OrderProcessorRepository.py OrderProcessorRepositoryImpl.py DBConnection.py Exception.py EcomApp.py
1 usage
90 def view_cart(self):
91     try:
92         connection = self.db_util.connection
93         cursor = connection.cursor()
94
95         customer_id = int(input("Enter customer ID to view cart: "))
96
97         query = "SELECT product_id, quantity FROM cart WHERE customer_id = %s"
98         cursor.execute(query, (customer_id,))
99         cart = cursor.fetchall()
100
101         print("Cart Contents:")
102         for product_id, quantity in cart:
103             print(f"Product ID: {product_id}, Quantity: {quantity}")
104
105         cursor.close()
106         connection.close()
107
108     except Exception as e:
109         print(f"Error viewing cart: {e}")
110
111 1 usage
112 def place_order(self):
113     try:
114         connection = self.db_util.connection
115         cursor = connection.cursor()
116
117         customer_id = int(input("Enter customer ID to place order: "))
118
119         products_quantities = []
120         while True:
121             product_id = int(input("Enter product ID to order (enter -1 to finish): "))
122             if product_id == -1:
123                 break
124             quantity = int(input("Enter quantity: "))
125             products_quantities.append((product_id, quantity))
126
127         query = "INSERT INTO orders (customer_id) VALUES (%s)"
128         cursor.execute(query, (customer_id,))
129         order_id = cursor.lastrowid
130
131         for product_id, quantity in products_quantities:
132             query = "INSERT INTO order_items (order_id, product_id, quantity) VALUES (%s, %s, %s)"
133             cursor.execute(query, (order_id, product_id, quantity))
134
135         connection.commit()
136         cursor.close()
137         connection.close()
138
139     except Exception as e:
140         print(f"Error placing order: {e}")
141
142 1 usage
143 if __name__ == "__main__":
144     app = EcomApp()
145     app.view_cart()
146     app.place_order()
147
148 EcomApp > place_order() > try
CaseStudy > Mainmode > Mainmodule > EcomApp.py 128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)
```

```
CaseStudy Version control EcomApp
ducts.py OrderItem.py OrderProcessorRepository.py OrderProcessorRepositoryImpl.py DBConnection.py Exception.py EcomApp.py
111 def place_order(self):
112     try:
113         connection = self.db_util.connection
114         cursor = connection.cursor()
115
116         customer_id = int(input("Enter customer ID to place order: "))
117
118         products_quantities = []
119         while True:
120             product_id = int(input("Enter product ID to order (enter -1 to finish): "))
121             if product_id == -1:
122                 break
123             quantity = int(input("Enter quantity: "))
124             products_quantities.append((product_id, quantity))
125
126         query = "INSERT INTO orders (customer_id) VALUES (%s)"
127         cursor.execute(query, (customer_id,))
128         order_id = cursor.lastrowid
129
130         for product_id, quantity in products_quantities:
131             query = "INSERT INTO order_items (order_id, product_id, quantity) VALUES (%s, %s, %s)"
132             cursor.execute(query, (order_id, product_id, quantity))
133
134         connection.commit()
135         cursor.close()
136         connection.close()
137
138     except Exception as e:
139         print(f"Error placing order: {e}")
140
141 1 usage
142 if __name__ == "__main__":
143     app = EcomApp()
144     app.view_cart()
145     app.place_order()
146
147 EcomApp > place_order() > try
CaseStudy > Mainmode > Mainmodule > EcomApp.py 128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)
```

```
CaseStudy Version control EcomApp
products.py OrderItem.py OrderProcessorRepository.py OrderProcessorRepositoryImpl.py DBConnection.py Exception.py EcomApp.py
134 connection.commit()
135 cursor.close()
136 connection.close()
137
138 print(f"Order placed successfully with ID: {order_id}")
139 except Exception as e:
140     print(f"Error placing order: {e}")
141
1 usage
142 def view_customer_order(self):
143     try:
144         connection = self.db_util.connection
145         cursor = connection.cursor()
146
147         customer_id = int(input("Enter customer ID to view orders: "))
148
149         query = "SELECT order_id, product_id, quantity FROM order_items WHERE order_id IN (SELECT order_id FROM orders WHERE customer_id = %s)"
150         cursor.execute(query, (customer_id,))
151         orders = cursor.fetchall()
152
153         print(f"Orders for Customer ID {customer_id}:")
154         for order in orders:
155             print(f"Order ID: {order[0]}, Product ID: {order[1]}, Quantity: {order[2]}")
156         cursor.close()
157         connection.close()
158
EcomApp > place_order() > try
CaseStudy > Mainmode > Mainmodule > EcomApp.py 128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)
```

```
CaseStudy Version control EcomApp
products.py OrderItem.py OrderProcessorRepository.py OrderProcessorRepositoryImpl.py DBConnection.py Exception.py EcomApp.py
159 except Exception as e:
160     print(f"Error viewing customer orders: {e}")
161
...
1 usage
162 def main(self):
163     while True:
164         print("E-commerce Application Menu:")
165         print("1. Register Customer")
166         print("2. Create Product")
167         print("3. Delete Product")
168         print("4. Add to Cart")
169         print("5. View Cart")
170         print("6. Place Order")
171         print("7. View Customer Order")
172         print("8. Exit")
173
174         choice = input("Enter your choice: ")
175
176         if choice == '1':
177             self.register_customer()
178         elif choice == '2':
179             self.create_product()
180         elif choice == '3':
181             self.delete_product()
182         elif choice == '4':
183             self.add_to_cart()
184         elif choice == '5':
185             self.view_cart()
186         elif choice == '6':
187             self.place_order()
188         elif choice == '7':
189             self.view_customer_order()
190         elif choice == '8':
191             break
192
EcomApp > place_order() > try
CaseStudy > Mainmode > Mainmodule > EcomApp.py 128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)
```

```
179         self.create_product()
180     elif choice == '3':
181         self.delete_product()
182     elif choice == '4':
183         self.add_to_cart()
184     elif choice == '5':
185         self.view_cart()
186     elif choice == '6':
187         self.place_order()
188     elif choice == '7':
189         self.view_customer_order()
190     elif choice == '8':
191         print('Exiting the application.')
192         break
193     else:
194         print('Invalid choice. Please choose a number from 1 to 8.')
195
196 if __name__ == '__main__':
197     app = EcomApp()
198     app.main()
199
```

EcomApp > place\_order() > try

CaseStudy > Mainmode > Mainmodule > EcomApp.py

128:40 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

Output:-

```
179         self.create_product()
180     elif choice == '3':
181         self.delete_product()
182     elif choice == '4':
183         self.add_to_cart()
184     elif choice == '5':
185         self.view_cart()
186     elif choice == '6':
187         self.place_order()
188     elif choice == '7':
189         self.view_customer_order()
190     elif choice == '8':
191         print('Exiting the application.')
192         break
193     else:
194         print('Invalid choice. Please choose a number from 1 to 8.')
195
196 if __name__ == '__main__':
197     app = EcomApp()
198     app.main()
199
```

EcomApp > main() > while True > else

Run EcomApp x DBConnection x

E-commerce Application Menu:  
1. Register Customer  
2. Create Product  
3. Delete Product  
4. Add to Cart  
5. View Cart  
6. Place Order  
7. View Customer Order  
8. Exit the application

CaseStudy > Mainmode > Mainmodule > EcomApp.py

194:77 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)

```
179         self.create_product()
180     elif choice == '3':
181         self.delete_product()
182     elif choice == '4':
183         self.add_to_cart()
184     elif choice == '5':
185         self.view_cart()
186     elif choice == '6':
187         self.place_order()
188     elif choice == '7':
189         self.view_customer_order()
190     elif choice == '8':
191         print("Exiting the application.")
192         break
193     else:
194         print("Invalid choice. Please choose a number from 1 to 8.")
EcomApp > main() > while True > else
```

Run EcomApp x DBConnection x

6. Place Order  
7. View Customer Order  
8. Exit  
Enter your choice: 1  
Enter customer ID: 12  
Enter customer name: munia  
Enter customer email: munia@email.com

```
179         self.create_product()
180     elif choice == '3':
181         self.delete_product()
182     elif choice == '4':
183         self.add_to_cart()
184     elif choice == '5':
185         self.view_cart()
186     elif choice == '6':
187         self.place_order()
188     elif choice == '7':
189         self.view_customer_order()
190     elif choice == '8':
191         print("Exiting the application.")
192         break
193     else:
194         print("Invalid choice. Please choose a number from 1 to 8.")
EcomApp > main() > while True > else
```

Run EcomApp x DBConnection x

1. register customer.  
2. Create Product.  
3. Delete Product.  
4. Add to cart.  
5. View cart.  
6. Place order.  
7. View Customer Order  
Customer registered successfully!  
E-commerce Application Menu:

```
12 | munia | munia@email.com | 10. 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. |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)
mysql>
```

## Unit Testing

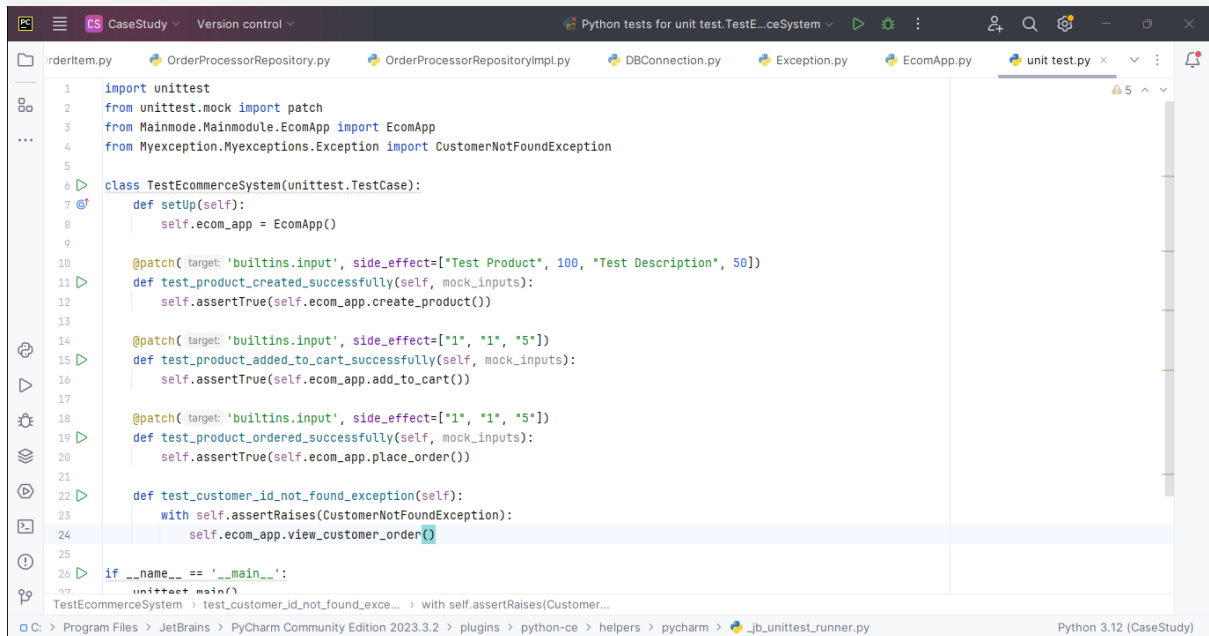
11. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system. Following questions to guide the creation of Unit test cases:

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.

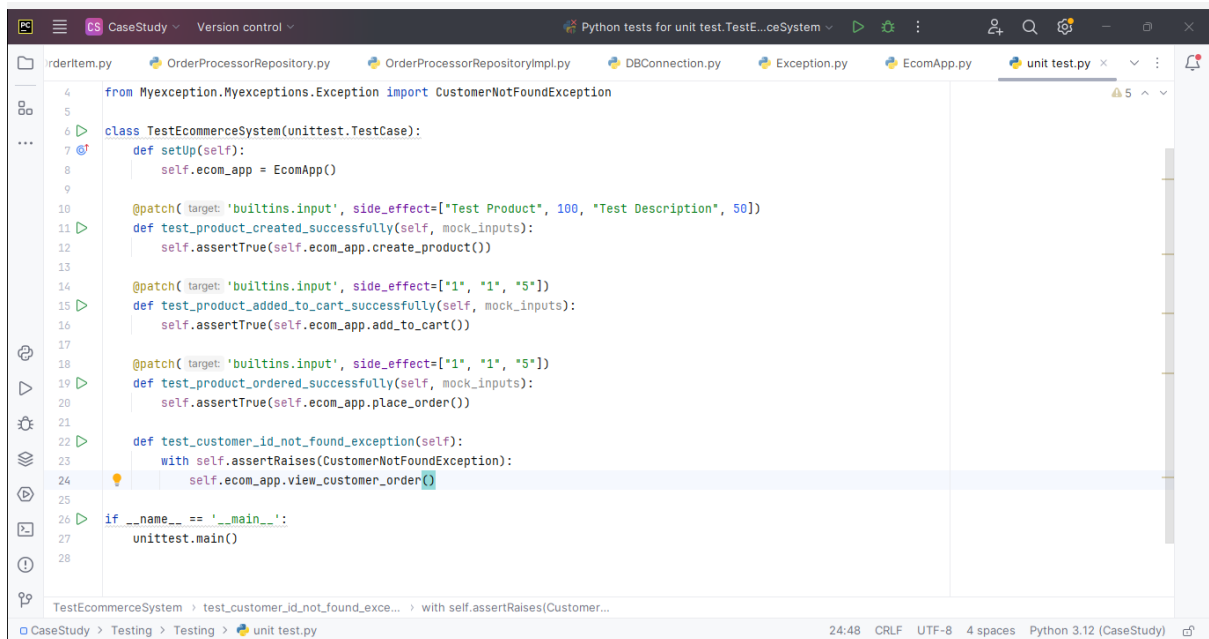
write test case to test exception is thrown correctly or not when customer id or product id not found in database.



```
1 import unittest
2 from unittest.mock import patch
3 from MainModule.MainModule.EcomApp import EcomApp
4 from Myexception.Myexceptions.Exception import CustomerNotFoundException
5
6 class TestEcommerceSystem(unittest.TestCase):
7     def setUp(self):
8         self.ecom_app = EcomApp()
9
10    @patch(target='builtins.input', side_effect=["Test Product", 100, "Test Description", 50])
11    def test_product_created_successfully(self, mock_inputs):
12        self.assertTrue(self.ecom_app.create_product())
13
14    @patch(target='builtins.input', side_effect=["1", "1", "5"])
15    def test_product_added_to_cart_successfully(self, mock_inputs):
16        self.assertTrue(self.ecom_app.add_to_cart())
17
18    @patch(target='builtins.input', side_effect=["1", "1", "5"])
19    def test_product_ordered_successfully(self, mock_inputs):
20        self.assertTrue(self.ecom_app.place_order())
21
22    def test_customer_id_not_found_exception(self):
23        with self.assertRaises(CustomerNotFoundException):
24            self.ecom_app.view_customer_order()
25
26 if __name__ == '__main__':
27     unittest.main()
```

TestEcommerceSystem > test\_customer\_id\_not\_found\_exce... > with self.assertRaises(Customer...

Python 3.12 (CaseStudy)



```
4 from Myexception.Myexceptions.Exception import CustomerNotFoundException
5
6 class TestEcommerceSystem(unittest.TestCase):
7     def setUp(self):
8         self.ecom_app = EcomApp()
9
10    @patch(target='builtins.input', side_effect=["Test Product", 100, "Test Description", 50])
11    def test_product_created_successfully(self, mock_inputs):
12        self.assertTrue(self.ecom_app.create_product())
13
14    @patch(target='builtins.input', side_effect=["1", "1", "5"])
15    def test_product_added_to_cart_successfully(self, mock_inputs):
16        self.assertTrue(self.ecom_app.add_to_cart())
17
18    @patch(target='builtins.input', side_effect=["1", "1", "5"])
19    def test_product_ordered_successfully(self, mock_inputs):
20        self.assertTrue(self.ecom_app.place_order())
21
22    def test_customer_id_not_found_exception(self):
23        with self.assertRaises(CustomerNotFoundException):
24            self.ecom_app.view_customer_order()
25
26 if __name__ == '__main__':
27     unittest.main()
```

TestEcommerceSystem > test\_customer\_id\_not\_found\_exce... > with self.assertRaises(Customer...

CaseStudy > Testing > Testing > unit test.py

24:48 CRLF UTF-8 4 spaces Python 3.12 (CaseStudy)