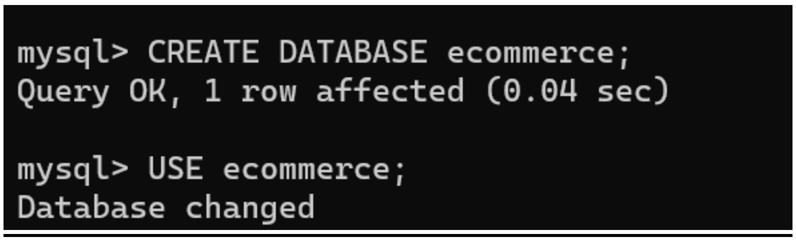
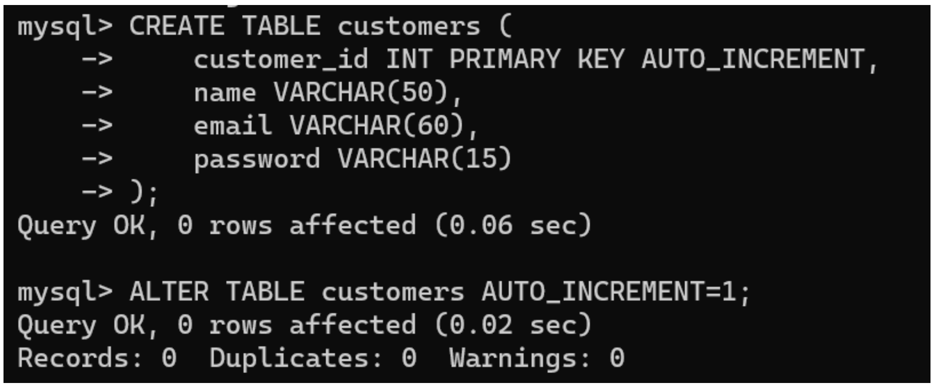
Case Study on E-Commerce Application

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



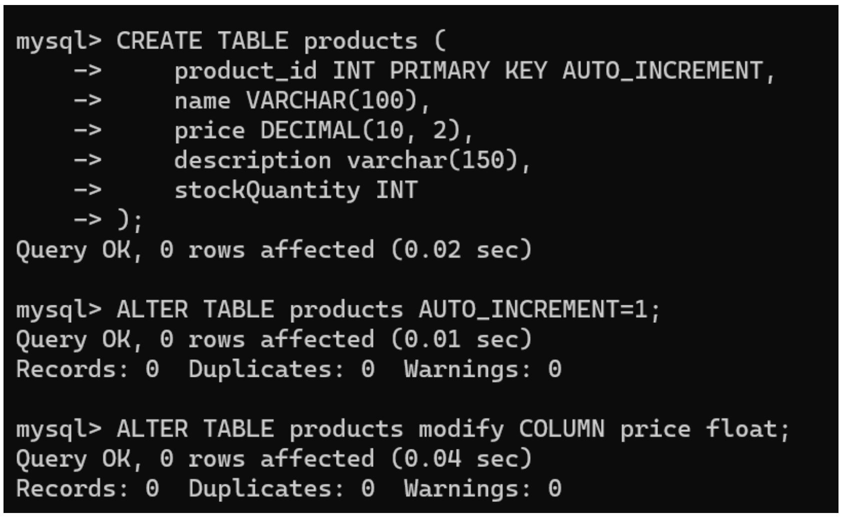
```
mysql> CREATE DATABASE ecommerce;
Query OK, 1 row affected (0.04 sec)

mysql> USE ecommerce;
Database changed
```

1. Schema Design:
   a. customers table:
      - customer_id (Primary Key)
      - name
      - email
      - password



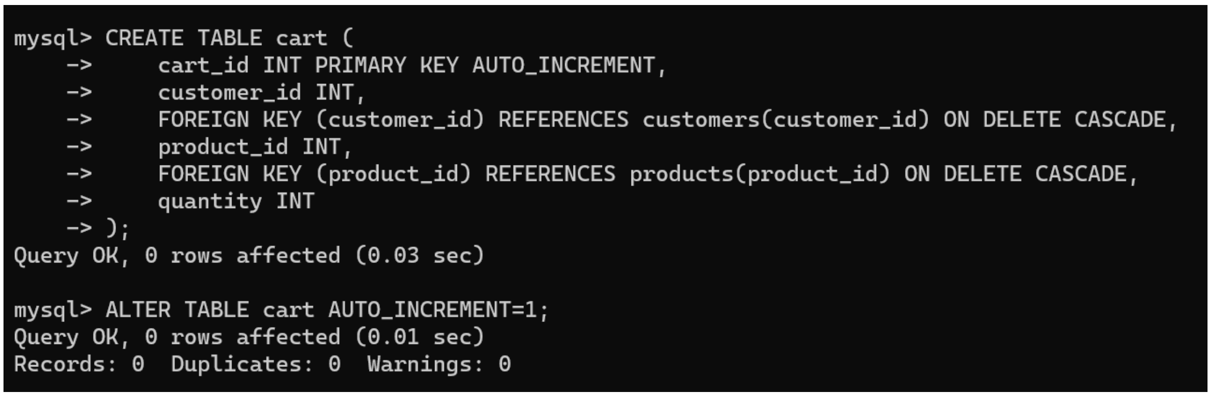
```
mysql> CREATE TABLE customers (
    ->     customer_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(50),
    ->     email VARCHAR(60),
    ->     password VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> ALTER TABLE customers AUTO_INCREMENT=1;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

   b. products table:
      - product_id (Primary Key)
      - name
      - price
      - description
      - stockQuantity



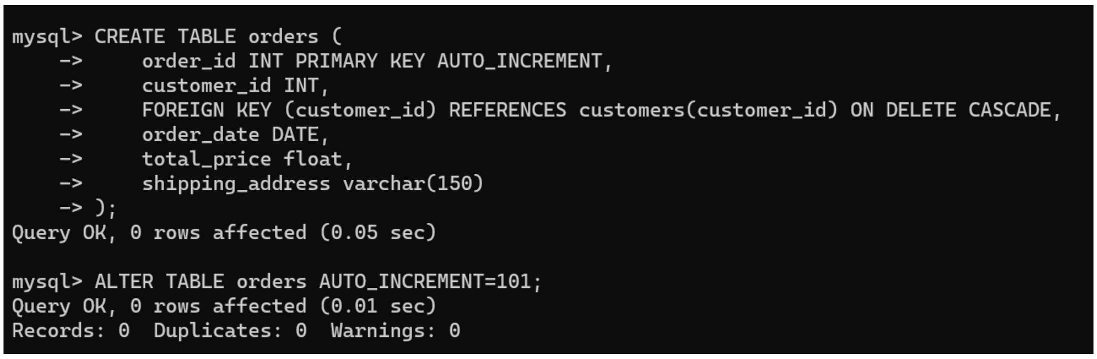
```
mysql> CREATE TABLE products (
    ->     product_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(100),
    ->     price DECIMAL(10, 2),
    ->     description varchar(150),
    ->     stockQuantity INT
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE products AUTO_INCREMENT=1;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE products modify COLUMN price float;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

c. cart table:
- cart_id (Primary Key)
- customer_id (Foreign Key)
- product_id (Foreign Key)
- quantity



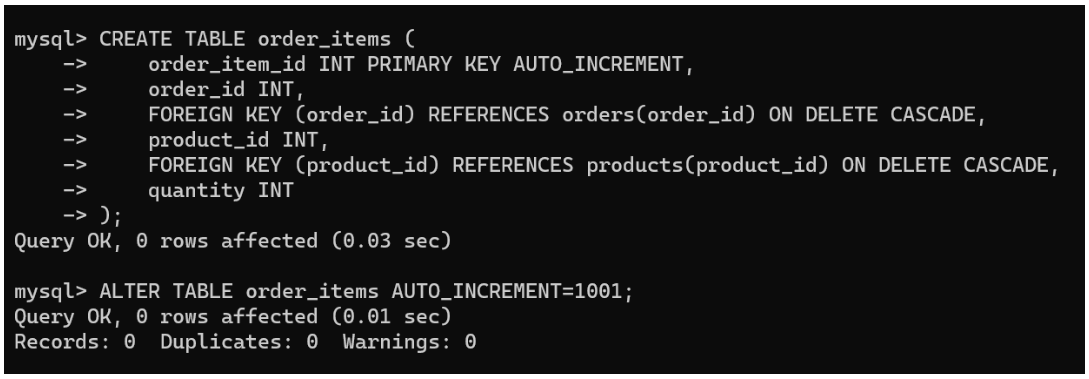
```
mysql> CREATE TABLE cart (
    ->     cart_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     customer_id INT,
    ->     FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE,
    ->     product_id INT,
    ->     FOREIGN KEY (product_id) REFERENCES products(product_id) ON DELETE CASCADE,
    ->     quantity INT
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE cart AUTO_INCREMENT=1;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

d. orders table:
- order_id (Primary Key)
- customer_id (Foreign Key)
- order_date
- total_price
- shipping_address



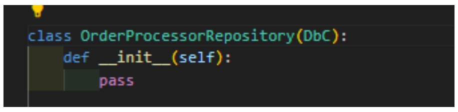
```
mysql> CREATE TABLE orders (
    ->     order_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     customer_id INT,
    ->     FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON DELETE CASCADE,
    ->     order_date DATE,
    ->     total_price float,
    ->     shipping_address varchar(150)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE orders AUTO_INCREMENT=101;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

e. order_items table (to store order details):
- order_item_id (Primary Key)
- order_id (Foreign Key)
- product_id (Foreign Key)
- quantity



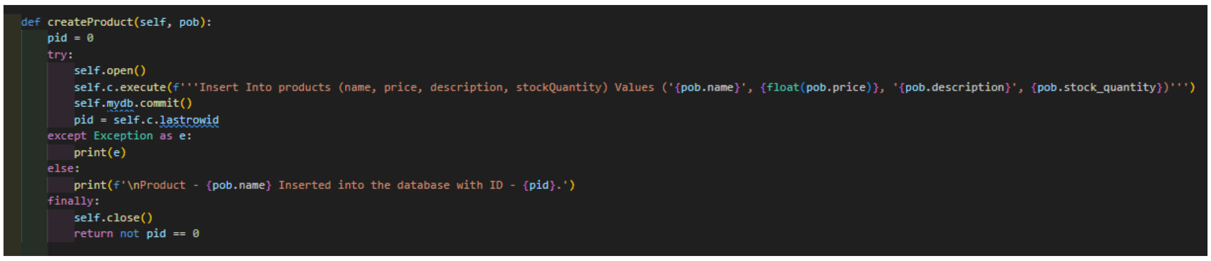
```
mysql> CREATE TABLE order_items (
    ->     order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     order_id INT,
    ->     FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
    ->     product_id INT,
    ->     FOREIGN KEY (product_id) REFERENCES products(product_id) ON DELETE CASCADE,
    ->     quantity INT
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE order_items AUTO_INCREMENT=1001;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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



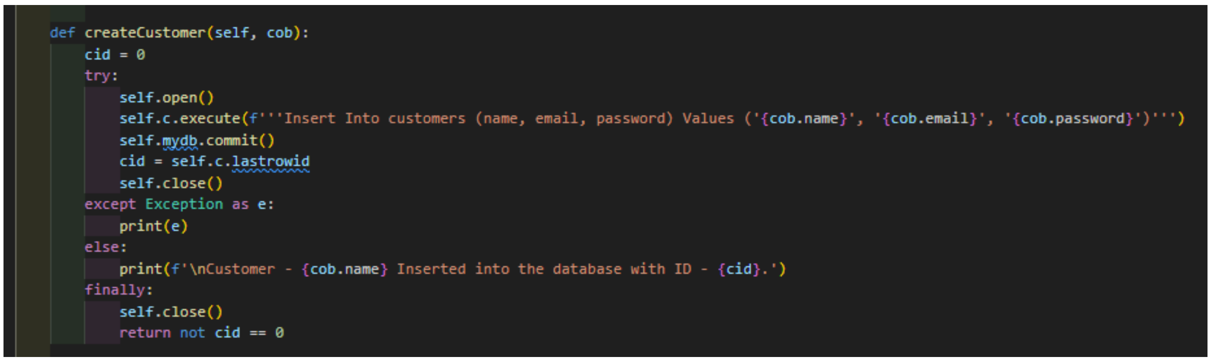
```
class OrderProcessorRepository(DbC):
    def __init__(self):
        pass
```

   a. createProduct()
      - parameter: Product
      - product return type: boolean



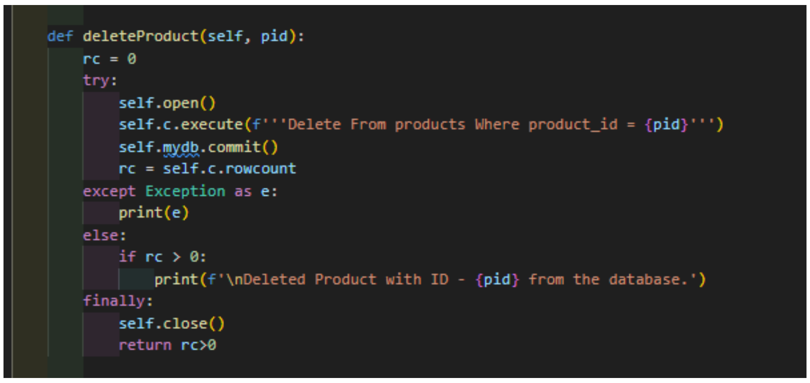
```
def createProduct(self, pob):
    pid = 0
    try:
        self.open()
        self.c.execute(f'''Insert Into products (name, price, description, stockQuantity) Values ('{pob.name}', {float(pob.price)}, '{pob.description}', {pob.stock_quantity})''')
        self.mydb.commit()
        pid = self.c.lastrowid
    except Exception as e:
        print(e)
    else:
        print(f'\nProduct - {pob.name} Inserted into the database with ID - {pid}.')
    finally:
        self.close()
        return not pid == 0
```

   b. createCustomer()
      - parameter: Customer
      - customer return type: boolean



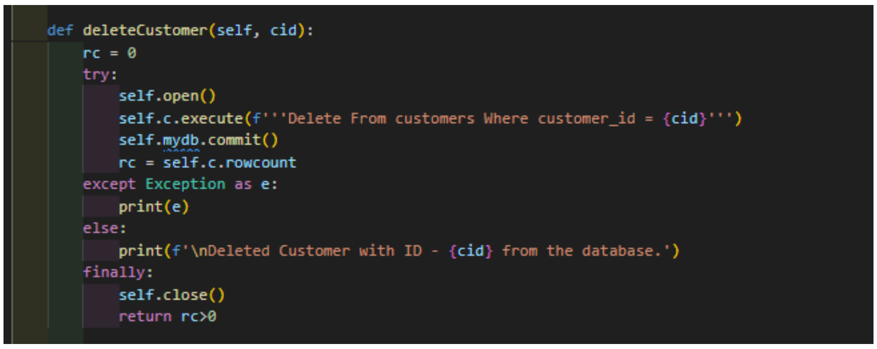
```
def createCustomer(self, cob):
    cid = 0
    try:
        self.open()
        self.c.execute(f'''Insert Into customers (name, email, password) Values ('{cob.name}', '{cob.email}', '{cob.password}')''')
        self.mydb.commit()
        cid = self.c.lastrowid
        self.close()
    except Exception as e:
        print(e)
    else:
        print(f'\nCustomer - {cob.name} Inserted into the database with ID - {cid}.')
    finally:
        self.close()
        return not cid == 0
```

   c. deleteProduct()
      - parameter: productId
      - return type: boolean



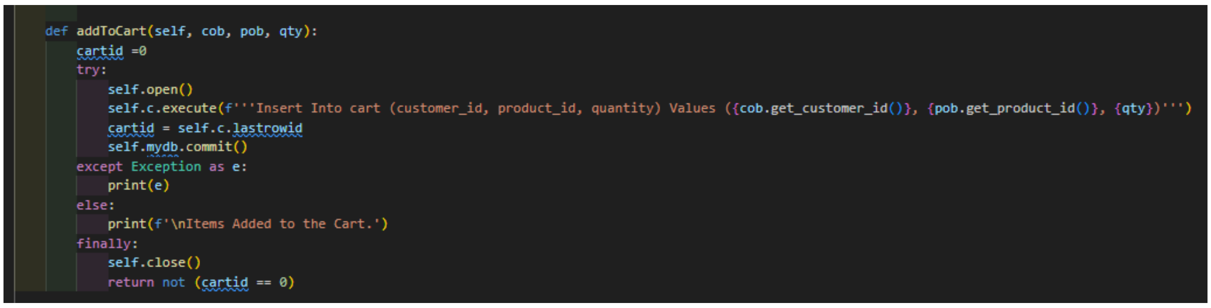
```
def deleteProduct(self, pid):
    rc = 0
    try:
        self.open()
        self.c.execute(f'''Delete From products Where product_id = {pid}''')
        self.mydb.commit()
        rc = self.c.rowcount
    except Exception as e:
        print(e)
    else:
        if rc > 0:
            print(f'\nDeleted Product with ID - {pid} from the database.')
    finally:
        self.close()
        return rc>0
```

d. deleteCustomer(customerId)
   - parameter: customerId
   - return type: boolean



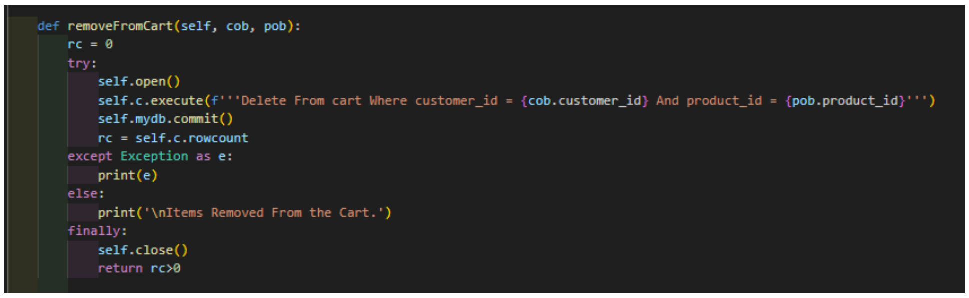
```python
def deleteCustomer(self, cid):
    rc = 0
    try:
        self.open()
        self.c.execute(f'''Delete From customers Where customer_id = {cid}''')
        self.mydb.commit()
        rc = self.c.rowcount
    except Exception as e:
        print(e)
    else:
        print(f'\nDeleted Customer with ID - {cid} from the database.')
    finally:
        self.close()
        return rc>0
```

e. addToCart(): insert the product in cart.
   - parameter: Customer customer, Product product, int quantity
   - return type: boolean



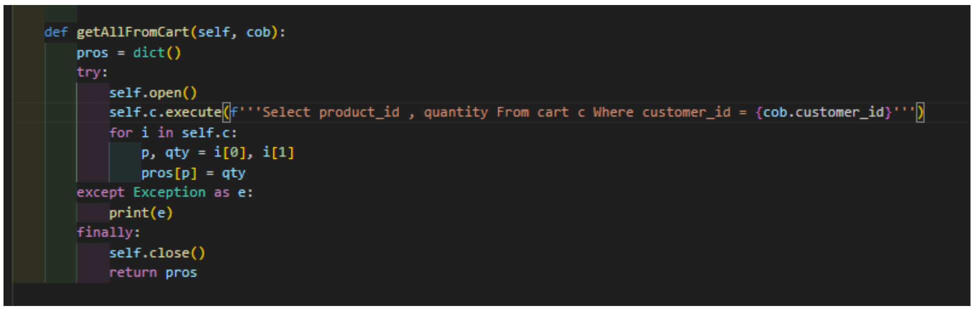
```python
def addToCart(self, cob, pob, qty):
    cartid =0
    try:
        self.open()
        self.c.execute(f'''Insert Into cart (customer_id, product_id, quantity) Values ({cob.get_customer_id()}, {pob.get_product_id()}, {qty})''')
        cartid = self.c.lastrowid
        self.mydb.commit()
    except Exception as e:
        print(e)
    else:
        print(f'\nItems Added to the Cart.')
    finally:
        self.close()
        return not (cartid == 0)
```

f. removeFromCart(): delete the product in cart.
   - parameter: Customer customer, Product product
   - return type: boolean



```python
def removeFromCart(self, cob, pob):
    rc = 0
    try:
        self.open()
        self.c.execute(f'''Delete From cart Where customer_id = {cob.customer_id} And product_id = {pob.product_id}''')
        self.mydb.commit()
        rc = self.c.rowcount
    except Exception as e:
        print(e)
    else:
        print('\nItems Removed From the Cart.')
    finally:
        self.close()
        return rc>0
```

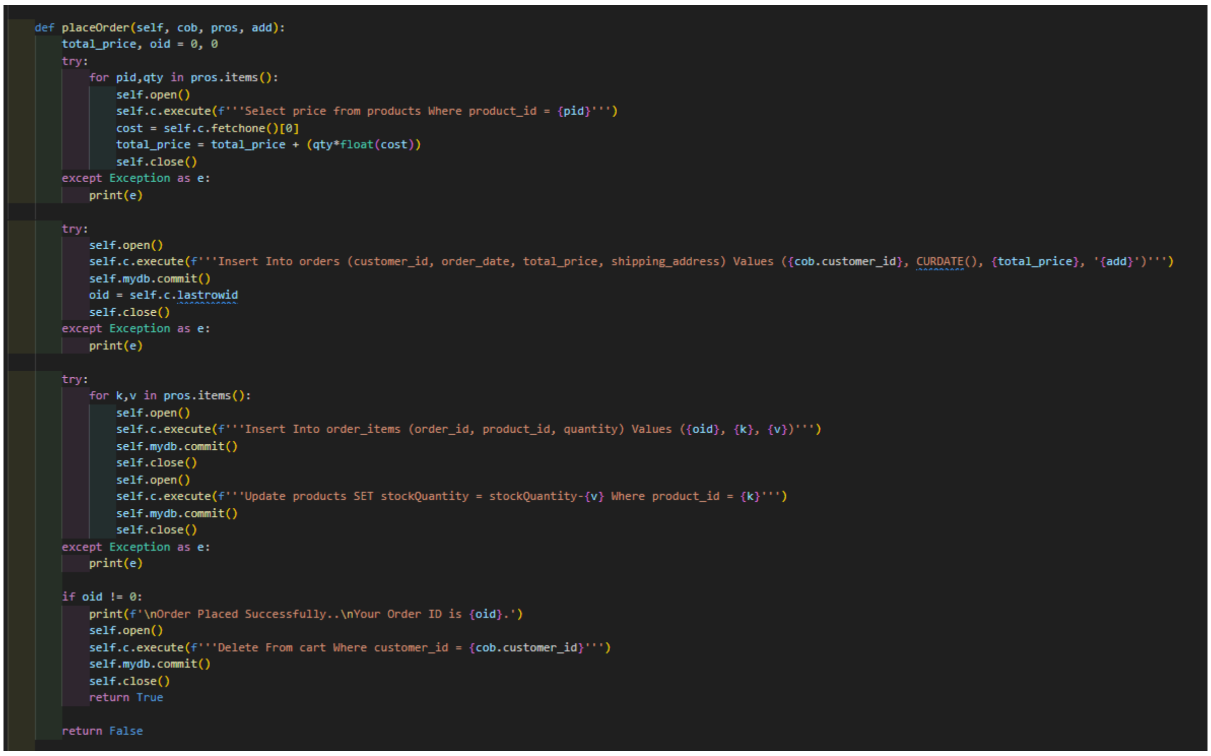
g. getAllFromCart(Customer customer): list the product in cart for a customer.
   - parameter: Customer customer
   - return type: list of product



```python
def getAllFromCart(self, cob):
    pros = dict()
    try:
        self.open()
        self.c.execute(f'''Select product_id , quantity From cart c Where customer_id = {cob.customer_id}''')
        for i in self.c:
            p, qty = i[0], i[1]
            pros[p] = qty
    except Exception as e:
        print(e)
    finally:
        self.close()
        return pros
```

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

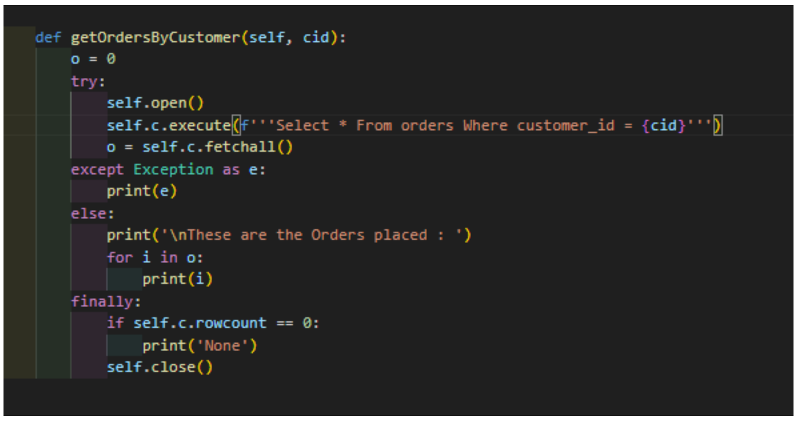
- parameter: Customer customer, list of product and quantity
- return type: boolean



```python
def placeOrder(self, cob, pros, add):
    total_price, oid = 0, 0
    try:
        for pid,qty in pros.items():
            self.open()
            self.c.execute(f'''Select price from products Where product_id = {pid}''')
            cost = self.c.fetchone()[0]
            total_price = total_price + (qty*float(cost))
            self.close()
    except Exception as e:
        print(e)

    try:
        self.open()
        self.c.execute(f'''Insert Into orders (customer_id, order_date, total_price, shipping_address) Values ({cob.customer_id}, CURDATE(), {total_price}, '{add}')''')
        self.mydb.commit()
        oid = self.c.lastrowid
        self.close()
    except Exception as e:
        print(e)

    try:
        for k,v in pros.items():
            self.open()
            self.c.execute(f'''Insert Into order_items (order_id, product_id, quantity) Values ({oid}, {k}, {v})''')
            self.mydb.commit()
            self.close()
            self.open()
            self.c.execute(f'''Update products SET stockQuantity = stockQuantity-{v} Where product_id = {k}''')
            self.mydb.commit()
            self.close()
    except Exception as e:
        print(e)

    if oid != 0:
        print(f'\nOrder Placed Successfully..\nYour Order ID is {oid}.')
        self.open()
        self.c.execute(f'''Delete From cart Where customer_id = {cob.customer_id}''')
        self.mydb.commit()
        self.close()
        return True

    return False
```

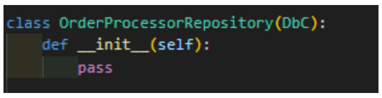
i. getOrdersByCustomer()

- parameter: customerid
- return type: list of product and quantity



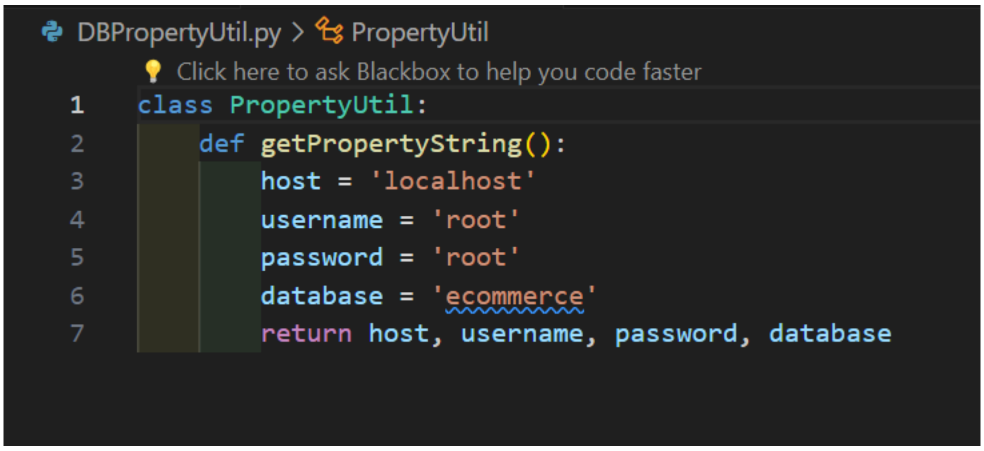
```python
def getOrdersByCustomer(self, cid):
    o = 0
    try:
        self.open()
        self.c.execute(f'''Select * From orders Where customer_id = {cid}''')
        o = self.c.fetchall()
    except Exception as e:
        print(e)
    else:
        print('\nThese are the Orders placed : ')
        for i in o:
            print(i)
    finally:
        if self.c.rowcount == 0:
            print('None')
        self.close()
```

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

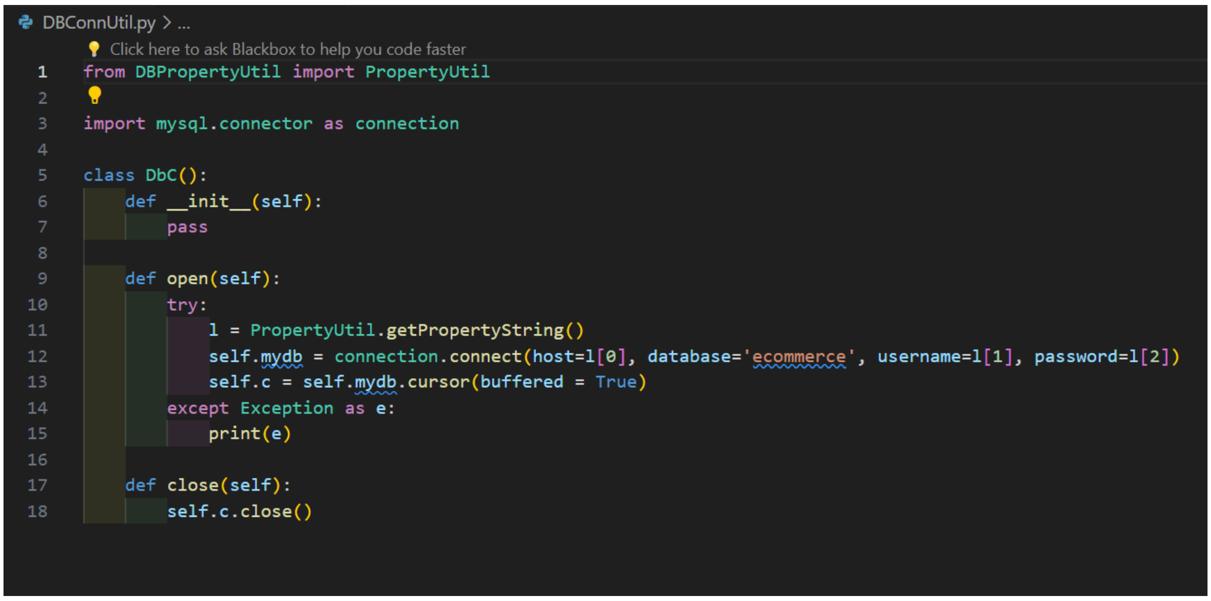


```python
class OrderProcessorRepository(DbC):
    def __init__(self):
        pass
```

3. 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.

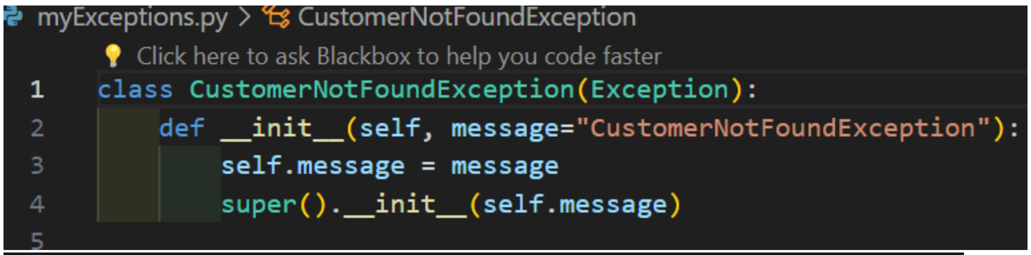

DBPropertyUtil.py > PropertyUtil
Click here to ask Blackbox to help you code faster

```python
class PropertyUtil:
    def getPropertyString():
        host = 'localhost'
        username = 'root'
        password = 'root'
        database = 'ecommerce'
        return host, username, password, database
```



```python
from DBPropertyUtil import PropertyUtil

import mysql.connector as connection

class DbC():
    def __init__(self):
        pass

    def open(self):
        try:
            l = PropertyUtil.getPropertyString()
            self.mydb = connection.connect(host=l[0], database='ecommerce', username=l[1], password=l[2])
            self.c = self.mydb.cursor(buffered = True)
        except Exception as e:
            print(e)

    def close(self):
        self.c.close()
```

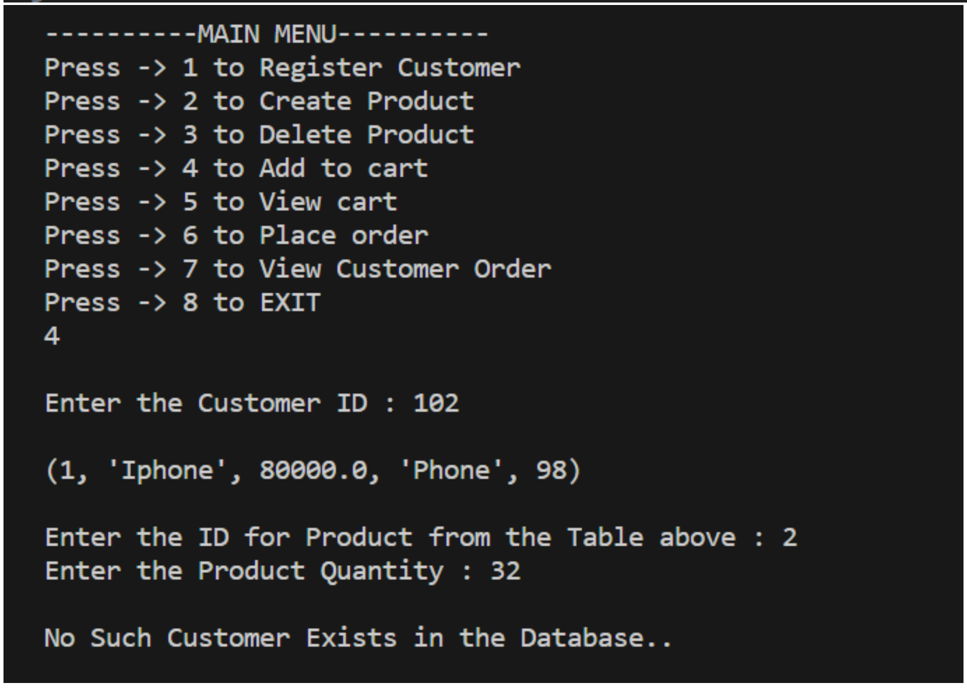
4. Create the exceptions in package myexceptions and create the following custom exceptions and throw them in methods whenever needed.
   Handle all the exceptions in main method:
   - CustomerNotFoundException: throw this exception when user enters an invalid customer id which doesn't exist in db



myExceptions.py > CustomerNotFoundException

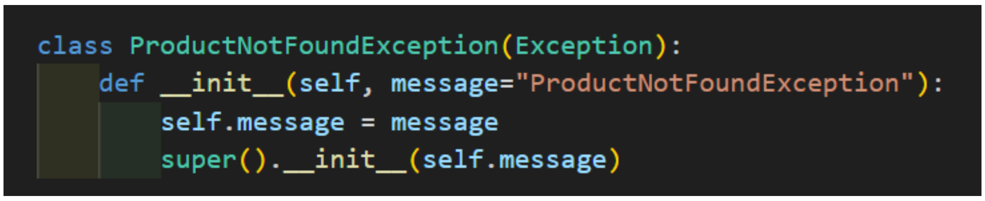
Click here to ask Blackbox to help you code faster

```python
class CustomerNotFoundException(Exception):
    def __init__(self, message="CustomerNotFoundException"):
        self.message = message
        super().__init__(self.message)

```

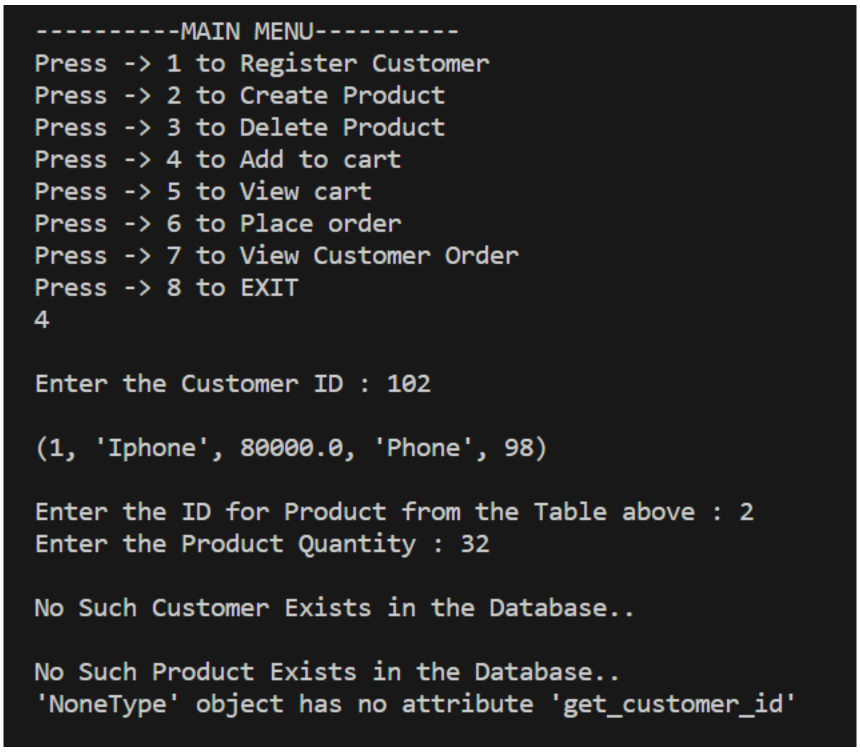


```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
4

Enter the Customer ID : 102

(1, 'Iphone', 80000.0, 'Phone', 98)

Enter the ID for Product from the Table above : 2
Enter the Product Quantity : 32

No Such Customer Exists in the Database..
```

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

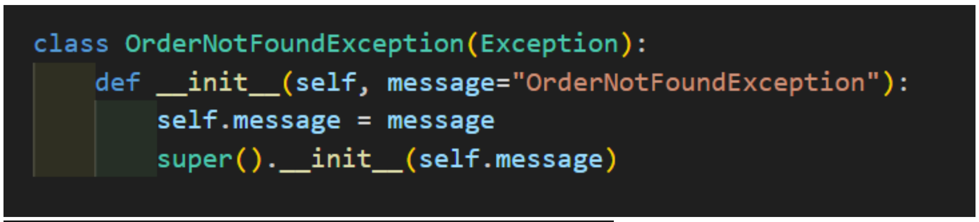


```python
class ProductNotFoundException(Exception):
    def __init__(self, message="ProductNotFoundException"):
        self.message = message
        super().__init__(self.message)
```

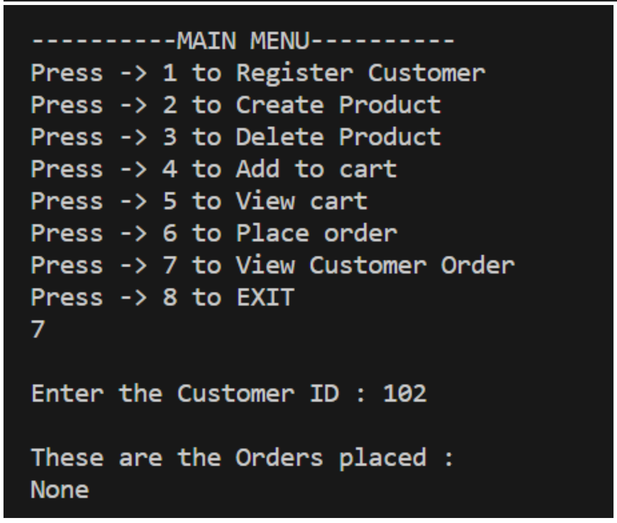


```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
4

Enter the Customer ID : 102

(1, 'Iphone', 80000.0, 'Phone', 98)

Enter the ID for Product from the Table above : 2
Enter the Product Quantity : 32

No Such Customer Exists in the Database..

No Such Product Exists in the Database..
'NoneType' object has no attribute 'get_customer_id'
```

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

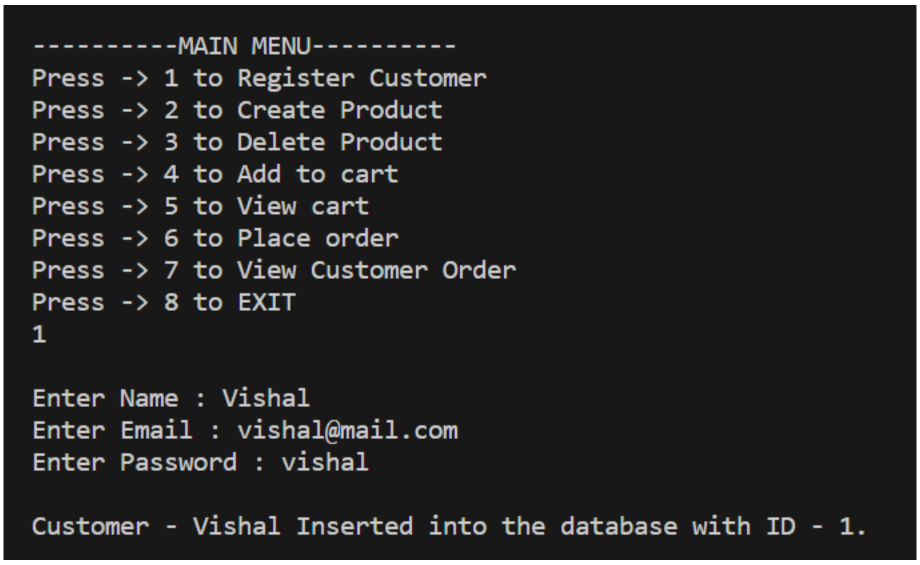


```python
class OrderNotFoundException(Exception):
    def __init__(self, message="OrderNotFoundException"):
        self.message = message
        super().__init__(self.message)
```



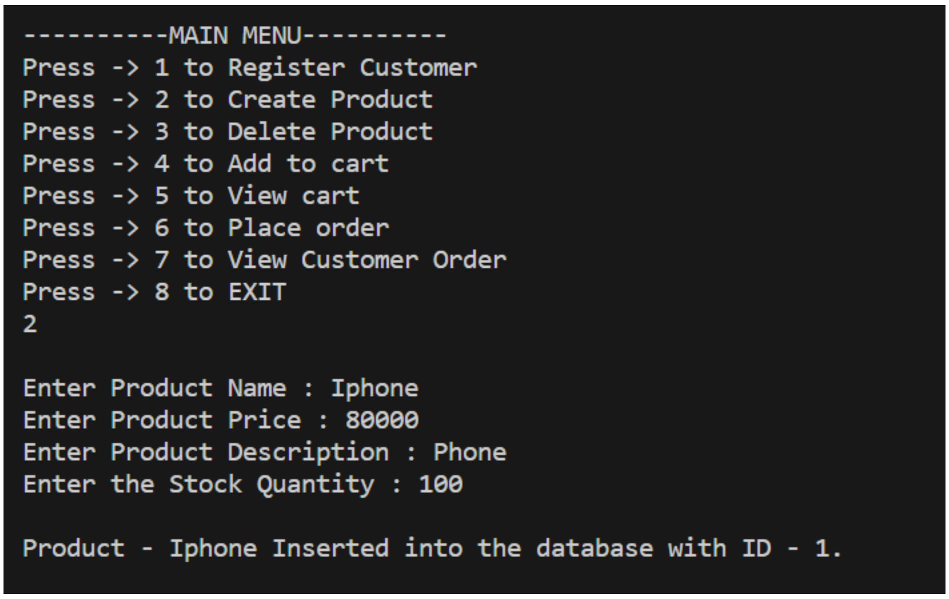
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
7

Enter the Customer ID : 102

These are the Orders placed :
None
```

5. 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.
    - Register Customer.



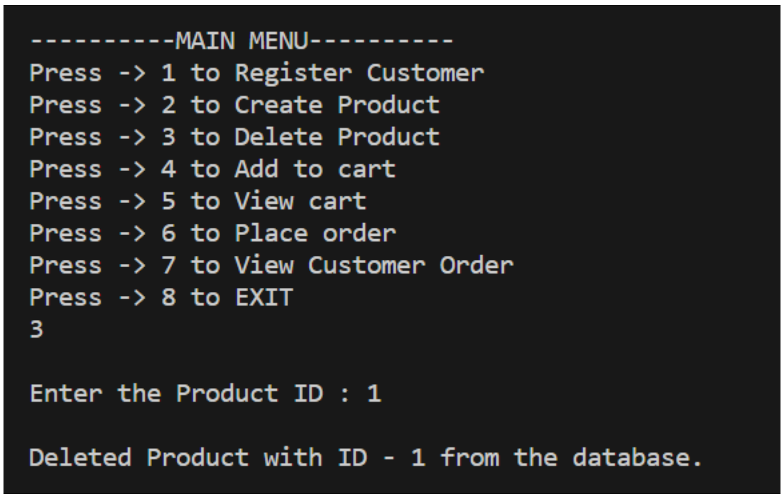
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
1

Enter Name : Vishal
Enter Email : vishal@mail.com
Enter Password : vishal

Customer - Vishal Inserted into the database with ID - 1.
```

    - Create Product.



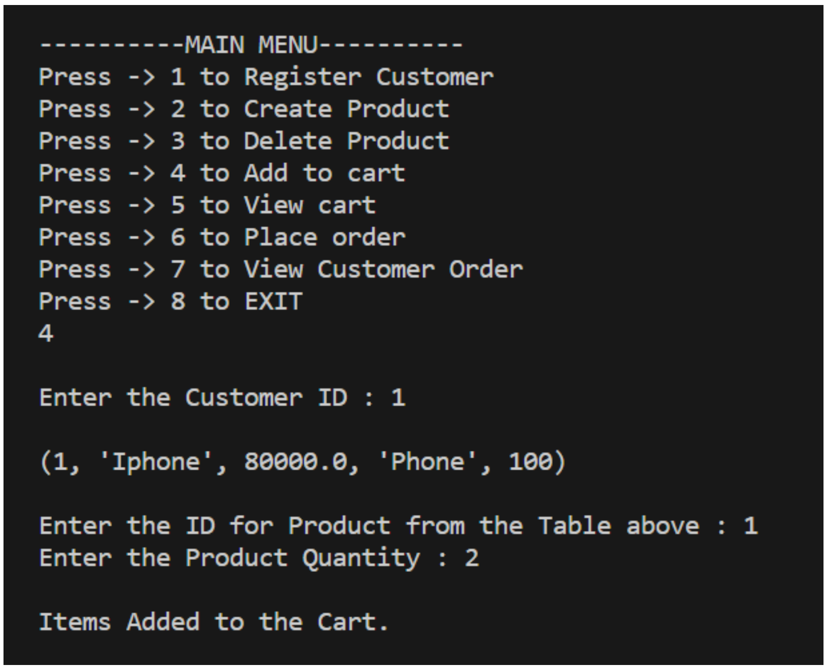
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
2

Enter Product Name : Iphone
Enter Product Price : 80000
Enter Product Description : Phone
Enter the Stock Quantity : 100

Product - Iphone Inserted into the database with ID - 1.
```

    - Delete Product.



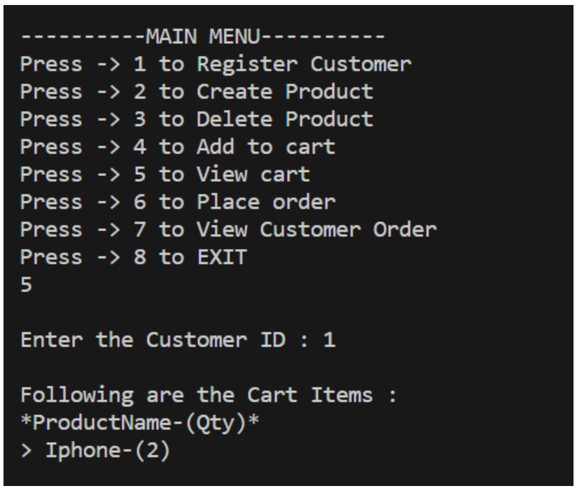
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
3

Enter the Product ID : 1

Deleted Product with ID - 1 from the database.
```

- Add to cart.



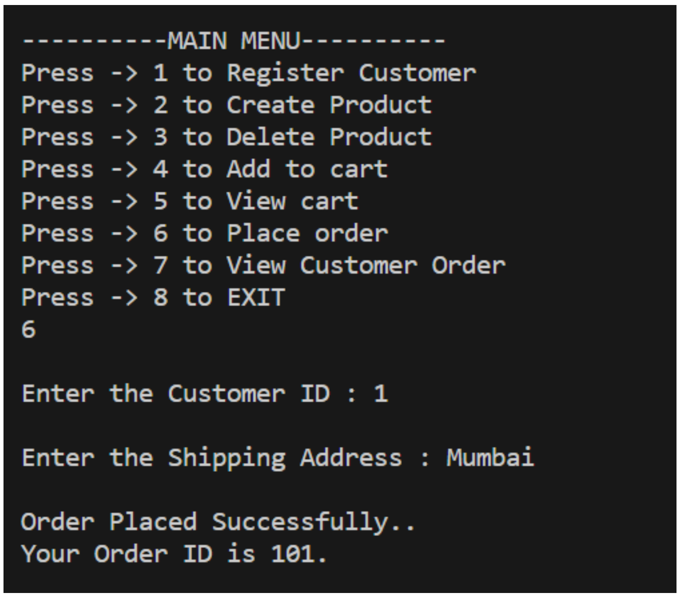
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
4

Enter the Customer ID : 1

(1, 'Iphone', 80000.0, 'Phone', 100)

Enter the ID for Product from the Table above : 1
Enter the Product Quantity : 2

Items Added to the Cart.
```

- View cart.



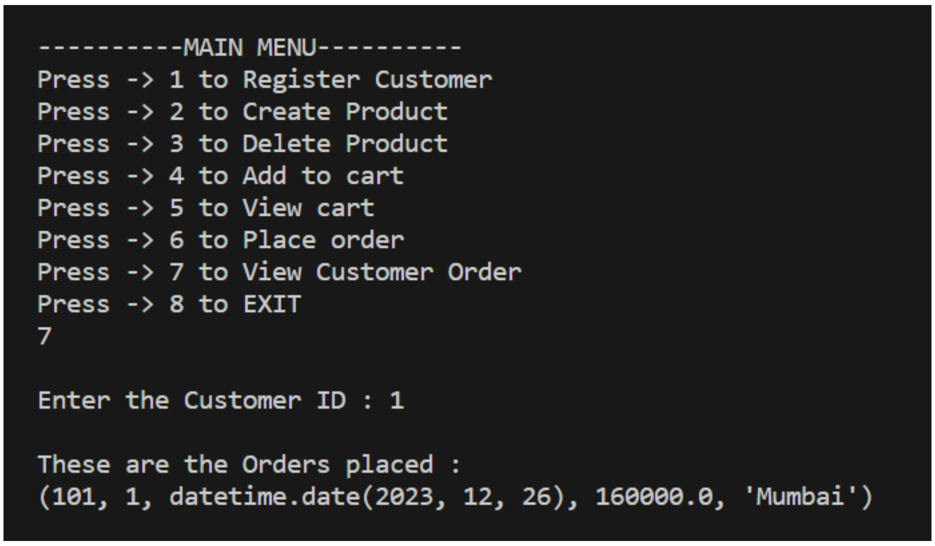
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
5

Enter the Customer ID : 1

Following are the Cart Items :
*ProductName-(Qty)*
> Iphone-(2)
```

- Place order.



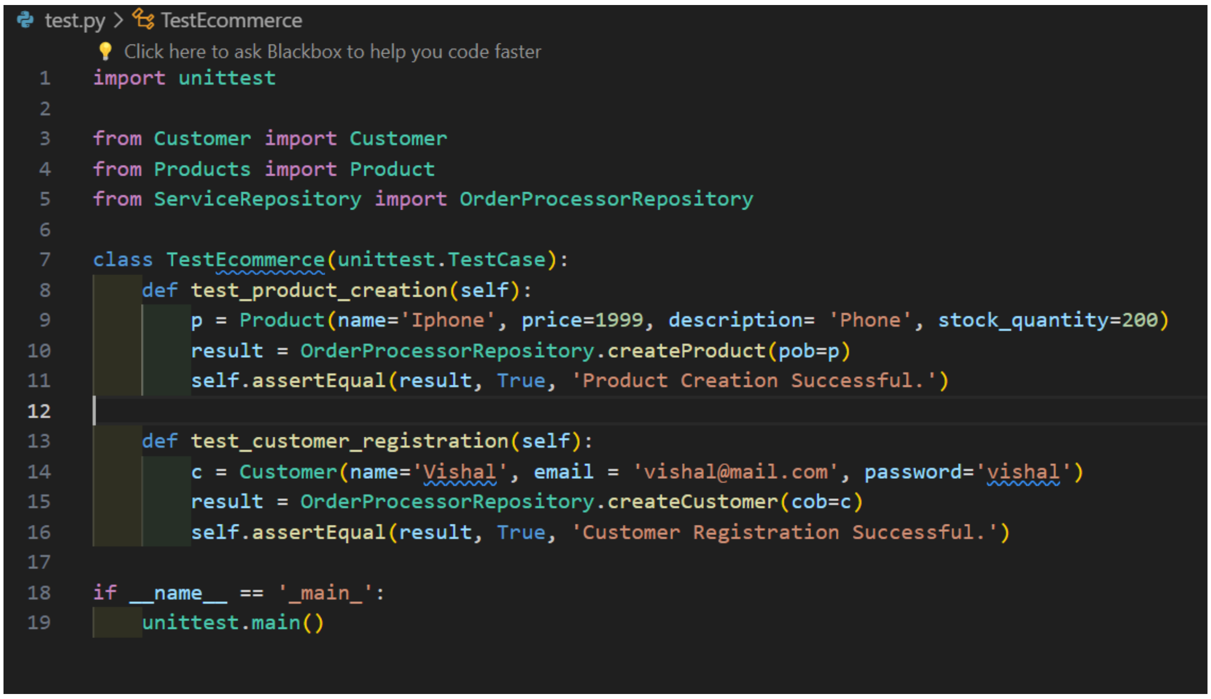
```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
6

Enter the Customer ID : 1

Enter the Shipping Address : Mumbai

Order Placed Successfully..
Your Order ID is 101.
```

- View Customer Order.



```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
7

Enter the Customer ID : 1

These are the Orders placed :
(101, 1, datetime.date(2023, 12, 26), 160000.0, 'Mumbai')
```

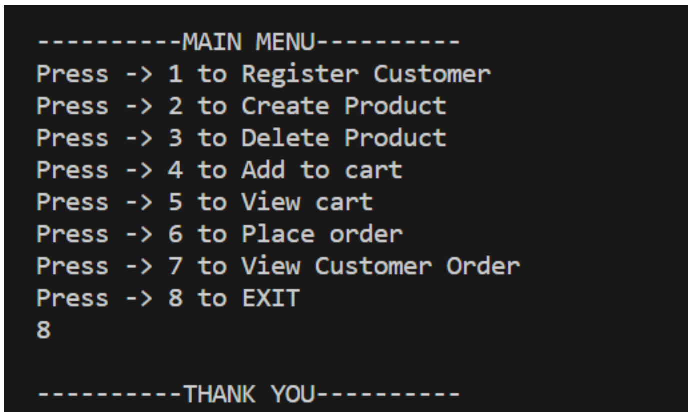
6. Create Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of your system.
   Following questions to guide the creation of Unit test cases: • Write test case to test Product created successfully or not.
   - Write test case to test product is added to cart 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.



test.py > TestEcommerce

Click here to ask Blackbox to help you code faster

```python
import unittest

from Customer import Customer
from Products import Product
from ServiceRepository import OrderProcessorRepository

class TestEcommerce(unittest.TestCase):
    def test_product_creation(self):
        p = Product(name='Iphone', price=1999, description= 'Phone', stock_quantity=200)
        result = OrderProcessorRepository.createProduct(pob=p)
        self.assertEqual(result, True, 'Product Creation Successful.')

    def test_customer_registration(self):
        c = Customer(name='Vishal', email = 'vishal@mail.com', password='vishal')
        result = OrderProcessorRepository.createCustomer(cob=c)
        self.assertEqual(result, True, 'Customer Registration Successful.')

if __name__ == '_main_':
    unittest.main()
```


```
----------MAIN MENU----------
Press -> 1 to Register Customer
Press -> 2 to Create Product
Press -> 3 to Delete Product
Press -> 4 to Add to cart
Press -> 5 to View cart
Press -> 6 to Place order
Press -> 7 to View Customer Order
Press -> 8 to EXIT
8

----------THANK YOU----------
```