

Case Study on Ecommerce Application

In the dao package, there exists a pivotal Service Provider Interface (SPI) or Abstract Class, termed OrderProcessorRepository, aimed at facilitating seamless interactions with the database pertaining to various aspects of the system, such as product and customer management, cart operations, and order processing. This interface lays out a clear set of methods, each serving a specific database-related function:

`createProduct(Product product)`: This method is designated for inserting a new product into the database. It returns a boolean value indicating the success of the operation.

`createCustomer(Customer customer)`: It is responsible for adding a new customer to the database. Like `createProduct`, it returns a boolean value based on the success of the operation.

`deleteProduct(String productId)`: This method enables the removal of a product from the database, with the product's ID serving as the identifier. Similar to the previous methods, it returns a boolean value to signal the success of the operation.

`deleteCustomer(String customerId)`: It facilitates the deletion of a customer from the database, utilizing the customer's ID for identification. As with other deletion methods, it returns a boolean value upon completion.

`addToCart(Customer customer, Product product, int quantity)`: This method allows the addition of a product to a customer's shopping cart along with the specified quantity. It returns a boolean value denoting the success of this action.

`removeFromCart(Customer customer, Product product)`: It supports the removal of a product from a customer's shopping cart. Once again, it returns a boolean value indicating the success or failure of this operation.

`getAllFromCart(Customer customer)`: This method retrieves all products currently residing in a customer's shopping cart, returning them as a list of products.

`placeOrder(Customer customer, List<OrderItem> orderItems, String shippingAddress)`: It orchestrates the process of placing an order for a customer, updating the respective order and order items tables in the database. The method returns a boolean value to signify the successful completion of this process.

`getOrdersByCustomer(String customerId)`: Finally, this method retrieves all orders associated with a specific customer, returning them as a list of orders.

`from abc import ABC, abstractmethod`

```
from typing import List, Dict, Tuple
from Entity.model import Product, Customer
```

```
class OrderProcessorRepository(ABC):
```

```
    @abstractmethod
```

```
    def create_product(self, product: Product) -> bool:
```

```
        pass
```

```
    @abstractmethod
```

```
    def create_customer(self, customer: Customer) -> bool:
```

```
        pass
```

```
    @abstractmethod
```

```
    def delete_product(self, product_id: int) -> bool:
```

```
        pass
```

```
    @abstractmethod
```

```
    def delete_customer(self, customer_id: int) -> bool:
```

```
        pass
```

```
    @abstractmethod
```

```
    def add_to_cart(self, customer_id: int, product_id: int, quantity: int) -> bool:
```

```
        pass
```

```
    @abstractmethod
```

```
    def remove_from_cart(self, customer_id: int, product_id: int) -> bool:
```

```
        pass
```

```
    @abstractmethod
```

```
    def get_all_from_cart(self, customer_id: int) -> List[Product]:
```

```
pass
```

```
@abstractmethod
```

```
def place_order(self, customer_id: int, products_quantities: List[Tuple[int, int]],  
shipping_address: str) -> bool:
```

```
pass
```

```
@abstractmethod
```

```
def get_orders_by_customer(self, customer_id: int) -> List[Tuple[Product, int]]:
```

```
pass
```

```
PS C:\Users\SNEHA BARANI KUMAR\OneDrive\Desktop\org_ecom> & "C:/Users/SNEHA BARANI KUMAR/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/SNEHA BARANI KUMAR/OneDrive/Desktop/org_ecom/ecom/dao/main.py"
```

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

```
import mysql.connector
```

```
from mysql.connector import Error
```

```
from typing import List, Tuple
```

```
from Entity.model import Customer, Product
```

```
class CustomerNotFoundException(Exception):
```

```
    def __init__(self, message="Customer not found."):
```

```
        self.message = message
```

```
        super().__init__(self.message)
```

```
class ProductNotFoundException(Exception):
```

```
    def __init__(self, message="Product not found."):
```

```
        self.message = message
```

```
        super().__init__(self.message)
```

```

class OrderNotFoundException(Exception):
    def __init__(self, message="Order not found."):
        self.message = message
        super().__init__(self.message)

class OrderProcessorRepositoryImpl():
    def __init__(self):
        self.connection = self.get_db_connection()

    def get_db_connection(self):
        connection = None
        try:
            connection = mysql.connector.connect(
                host='localhost',
                user='root',
                password='Sneha@2705',
                database='ecom',
                port="3306"
            )
            if connection.is_connected():
                print("Connected to MySQL database")
        except Error as e:
            print(f"Error connecting to MySQL: {e}")
        return connection

    def createProduct(self, product: Product) -> bool:
        try:
            with self.connection.cursor() as cursor:
                sql = "INSERT INTO products (product_id, name, price, description, stockQuantity) VALUES (%s, %s, %s, %s, %s)"
                cursor.execute(sql, (product.product_id, product.name, product.price, product.description, product.stock_quantity))

```

```

        self.connection.commit()

        print("Product created successfully.")

        return True

    except Error as e:

        print(f"Error creating product: {e}")

        return False

    def createCustomer(self, customer: Customer) -> bool:

        try:

            with self.connection.cursor() as cursor:

                sql = "INSERT INTO customers (customer_id, name, email, password,address)
VALUES (%s, %s, %s, %s, %s)"

                cursor.execute(sql, (customer.customer_id,customer.name, customer.email,
customer.password, customer.address))

                self.connection.commit()

                print("Customer created successfully.")

                return True

            except Error as e:

                print(f"Error creating customer: {e}")

                return False

        def deleteProduct(self, product_id: int) -> bool:

            try:

                with self.connection.cursor() as cursor:

                    sql = "DELETE FROM products WHERE product_id = %s"

                    cursor.execute(sql, (product_id,))

                    self.connection.commit()

                    print("Product deleted successfully.")

                    return True

                except Error as e:

                    print(f"Error deleting product: {e}")

                    return False

```

```
def deleteCustomer(self, customer_id: int) -> bool:
```

```
    try:
```

```
        with self.connection.cursor() as cursor:
```

```
            sql = "DELETE FROM customers WHERE customer_id = %s"
```

```
            cursor.execute(sql, (customer_id,))
```

```
        self.connection.commit()
```

```
        print("Customer deleted successfully.")
```

```
        return True
```

```
    except Error as e:
```

```
        print(f"Error deleting customer: {e}")
```

```
        return False
```

```
def addToCart(self, customer_id: int, product_id: int, quantity: int) -> bool:
```

```
    try:
```

```
        with self.connection.cursor() as cursor:
```

```
            sql = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s,  
%s)"
```

```
            cursor.execute(sql, (customer_id, product_id, quantity))
```

```
        self.connection.commit()
```

```
        print("Product added to cart successfully.")
```

```
        return True
```

```
    except Error as e:
```

```
        print(f"Error adding product to cart: {e}")
```

```
        return False
```

```
def removeFromCart(self, customer_id: int, product_id: int) -> bool:
```

```
    try:
```

```
        with self.connection.cursor() as cursor:
```

```
            sql = "DELETE FROM cart WHERE customer_id = %s AND product_id = %s"
```

```
            cursor.execute(sql, (customer_id, product_id))
```

```
        self.connection.commit()
```

```

        print("Product removed from cart successfully.")
        return True
    except Error as e:
        print(f"Error removing product from cart: {e}")
        return False

def getAllFromCart(self, customer_id: int) -> List[Product]:
    try:
        with self.connection.cursor() as cursor:
            sql = "SELECT * FROM products WHERE product_id IN (SELECT product_id
FROM cart WHERE customer_id = %s)"
            cursor.execute(sql, (customer_id,))
            products = cursor.fetchall()
            return [Product(**product) for product in products]
    except Error as e:
        print(f"Error retrieving products from cart: {e}")
        return []

def placeOrder(self, customer_id: int, products_quantities: List[Tuple[int, int]],
shipping_address: str) -> bool:
    try:
        with self.connection.cursor() as cursor:
            order_sql = "INSERT INTO orders (customer_id, order_date, shipping_address)
VALUES (%s, NOW(), %s)"
            cursor.execute(order_sql, (customer_id, shipping_address))
            order_id = cursor.lastrowid
            order_item_sql = "INSERT INTO order_items (order_id, product_id, quantity)
VALUES (%s, %s, %s)"

            for product_id, quantity in products_quantities:
                cursor.execute(order_item_sql, (order_id, product_id, quantity))

```

```
        self.connection.commit()

        print("Order placed successfully.")

        return True

    except Error as e:

        print(f"Error placing order: {e}")

        self.connection.rollback()

        return False
```

```
def getOrdersByCustomer(self, customer_id: int) -> List[Tuple[Product, int]]:
```

```
    try:

        with self.connection.cursor() as cursor:

            sql = """SELECT p.*, oi.quantity

                        FROM order_items oi

                        JOIN products p ON oi.product_id = p.product_id

                        JOIN orders o ON oi.order_id = o.order_id

                        WHERE o.customer_id = %s"""

            cursor.execute(sql, (customer_id,))

            order_items = cursor.fetchall()

            if order_items:

                return [(Product(**item), item['quantity']) for item in order_items]

            else:

                raise OrderNotFoundException("Order not found for the given customer.")

    except Error as e:

        print(f"Error retrieving orders by customer: {e}")

        return []
```

```
def main():
```

```
    while True:

        print("\nChoose an operation:")

        print("1. Create Product")
```



```
print("2. Create Customer")
print("3. Delete Product")
print("4. Delete Customer")
print("5. Add to Cart")
print("6. Remove from Cart")
print("7. View Cart")
print("8. Place Order")
print("9. Get Orders By Customer")
print("10. Exit")
```

```
choice = input("Enter your choice: ")
```

```
if choice == '1':
```

```
    prod_id = int(input("Enter product ID: "))
    product_name = input("Enter product name: ")
    product_price = float(input("Enter product price: "))
    product_description = input("Enter product description: ")
    product_quantity = int(input("Enter product quantity: "))
    product = Product(product_id=prod_id, name=product_name, price=product_price,
description=product_description, stock_quantity=product_quantity)
    order_processor.createProduct(product)
```

```
elif choice == '2':
```

```
    customer_id = int(input("Enter Customer Id : "))
    customer_name = input("Enter customer name: ")
    customer_email = input("Enter customer email: ")
    customer_password = input("Enter customer password: ")
    address = input("Enter Customer Address : ")
    customer = Customer(customer_id = customer_id, name=customer_name,
email=customer_email, password=customer_password, address=address)
    order_processor.createCustomer(customer)
```

```
elif choice == '3':  
    product_id = int(input("Enter product ID to delete: "))  
    order_processor.deleteProduct(product_id)  
  
elif choice == '4':  
    customer_id = int(input("Enter customer ID to delete: "))  
    order_processor.deleteCustomer(customer_id)  
  
elif choice == '5':  
    customer_id = int(input("Enter customer ID: "))  
    product_id = int(input("Enter product ID to add to cart: "))  
    quantity = int(input("Enter quantity: "))  
    order_processor.addToCart(customer_id, product_id, quantity)  
  
elif choice == '6':  
    customer_id = int(input("Enter customer ID: "))  
    product_id = int(input("Enter product ID to remove from cart: "))  
    order_processor.removeFromCart(customer_id, product_id)  
  
elif choice == '7':  
    customer_id = int(input("Enter customer ID: "))  
    cart_items = order_processor.getAllFromCart(customer_id)  
    print("Items in cart:")  
    for item in cart_items:  
        print(item.name)  
  
elif choice == '8':  
    customer_id = int(input("Enter customer ID: "))  
    shipping_address = input("Enter shipping address: ")
```

```
products_quantities = []

while True:

    product_id = int(input("Enter product ID (0 to stop): "))

    if product_id == 0:

        break

    quantity = int(input("Enter quantity: "))

    products_quantities.append((product_id, quantity))

    order_processor.placeOrder(customer_id, products_quantities, shipping_address)

elif choice == '9':

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

    orders = order_processor.getOrdersByCustomer(customer_id)

    print("Orders by customer:")

    for order in orders:

        print(order[0].name, "-", order[1], "quantity")

elif choice == '10':

    print("Exiting...")

    break

else:

    print("Invalid choice. Please enter a number between 1 and 10.")

if __name__ == "__main__":

    order_processor = OrderProcessorRepositoryImpl()

    main()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

6. Place Order
7. View Customer Order
8. Exit
Enter your choice: 1
Enter Customer_id : 123
Enter customer name: sneha
Enter customer email: Sneha@exampl.ecom
Enter customer password: qwertyuiop
Enter Customer's Address : oiuytre
Customer created successfully.
Customer registered successfully.
```

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.

```
import mysql.connector
from util.property import PropertyUtil
```

```
class DBConnection:
    connection = None

    @staticmethod
    def getConnection():
```



```
properties[key.strip()] = value.strip()

return properties
```

```
PS C:\Users\SNEHA BARANI KUMAR\OneDrive\Desktop\org_ecom> & "c:/Users/SNEHA BARANI KUMAR/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/SNEHA BARANI KUMAR/OneDrive/Desktop/org_ecom/ecom/dao/main.py"
```

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

```
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found."):
        self.message = message
        super().__init__(self.message)
```

```
class ProductNotFoundException(Exception):
    def __init__(self, message="Product not found."):
        self.message = message
        super().__init__(self.message)
```

```
class OrderNotFoundException(Exception):
    def __init__(self, message="Order not found."):
        self.message = message
        super().__init__(self.message)
```

```
Enter your choice: & "c:/Users/SNEHA BARANI KUMAR/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/SNEHA BARANI KUMAR/OneDrive/Desktop/org_ecom/ecom/dao/myexception.py"
```

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 OrderProcessorRepositoryImpl import OrderProcessorRepositoryImpl
```

```
from Entity.model import Customer, Product
```

```
class CustomerNotFoundException(Exception):
```

```
    def __init__(self, message="Customer not found."):
```

```
        self.message = message
```

```
        super().__init__(self.message)
```

```
class ProductNotFoundException(Exception):
```

```
    def __init__(self, message="Product not found."):
```

```
        self.message = message
```

```
        super().__init__(self.message)
```

```
class OrderNotFoundException(Exception):
```

```
    def __init__(self, message="Order not found."):
```

```
        self.message = message
```

```
        super().__init__(self.message)
```

```
class EcomApp:
```

```
    def __init__(self):
```

```
        self.order_processor = OrderProcessorRepositoryImpl()
```

```
def main(self):
    while True:
        print("\nChoose an operation:")
        print("1. Register Customer")
        print("2. Create Product")
        print("3. Delete Product")
        print("4. Add to Cart")
        print("5. View Cart")
        print("6. Place Order")
        print("7. View Customer Order")
        print("8. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            self.register_customer()
        elif choice == '2':
            self.create_product()
        elif choice == '3':
            self.delete_product()
        elif choice == '4':
            self.add_to_cart()
        elif choice == '5':
            self.view_cart()
        elif choice == '6':
            self.place_order()
        elif choice == '7':
            self.view_customer_order()
        elif choice == '8':
            print("Exiting...")
```



```

        break

    else:

        print("Invalid choice. Please enter a number between 1 and 8.")

def register_customer(self):
    customer_id=int(input("Enter Customer_id : "))
    name = input("Enter customer name: ")
    email = input("Enter customer email: ")
    password = input("Enter customer password: ")
    address = input("Enter Customer's Address : ")
    customer = Customer(customer_id, name, email, password,address)
    if self.order_processor.createCustomer(customer):
        print("Customer registered successfully.")
    else:
        print("Failed to register customer.")

def create_product(self):
    product_id = int(input("Enter product id : "))
    name = input("Enter product name: ")
    price = float(input("Enter product price: "))
    description = input("Enter product description: ")
    stock_quantity = int(input("Enter product stock quantity: "))
    product = Product(product_id, name, price, description, stock_quantity)
    if self.order_processor.createProduct(product):
        print("Product created successfully.")
    else:
        print("Failed to create product.")

def delete_product(self):
    product_id = int(input("Enter product ID to delete: "))

```

```
if self.order_processor.deleteProduct(product_id):
    print("Product deleted successfully.")
else:
    print("Failed to delete product.")

def add_to_cart(self):
    customer_id = int(input("Enter customer ID: "))
    product_id = int(input("Enter product ID to add to cart: "))
    quantity = int(input("Enter quantity: "))
    try:
        if self.order_processor.addToCart(customer_id, product_id, quantity):
            print("Product added to cart successfully.")
        else:
            print("Failed to add product to cart.")
    except (CustomerNotFoundException, ProductNotFoundException) as e:
        print(e)

def view_cart(self):
    customer_id = int(input("Enter customer ID to view cart: "))
    try:
        cart_items = self.order_processor.getAllFromCart(customer_id)
        print("Cart Items:")
        for item in cart_items:
            print(f'{item["product"].name} - Quantity: {item["quantity"]}')
    except CustomerNotFoundException as e:
        print(e)

def place_order(self):
    customer_id = int(input("Enter customer ID to place order: "))
    shipping_address = input("Enter shipping address: ")
```

```

products_quantities = []
while True:
    product_id = int(input("Enter product ID (0 to stop): "))
    if product_id == 0:
        break
    quantity = int(input("Enter quantity: "))
    products_quantities.append((product_id, quantity))
    try:
        if self.order_processor.placeOrder(customer_id, products_quantities,
shipping_address):
            print("Order placed successfully.")
        else:
            print("Failed to place order.")
    except CustomerNotFoundException as e:
        print(e)

def view_customer_order(self):
    customer_id = int(input("Enter customer ID to view orders: "))
    try:
        orders = self.order_processor.getOrdersByCustomer(customer_id)
        print("Customer Orders:")
        for order in orders:
            print(
                f"Order ID: {order['order_id']}, Total Price: {order['total_price']}, Order Date:
{order['order_date']}")
    except CustomerNotFoundException as e:
        print(e)

if __name__ == "__main__":
    App = EcomApp()
    App.main()

```

```
Choose an operation:
1. Create Product
2. Create Customer
3. Delete Product
4. Delete Customer
5. Add to Cart
6. Remove from Cart
7. View Cart
8. Place Order
9. Get Orders By Customer
10. Exit
Enter your choice: 10
Exiting...
Connected to MySQL database
```

```
Choose an operation:
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
Enter your choice: 1
Enter Customer_id : 123
Enter customer name: sneha
Enter customer email: Sneha@exampl.ecom
Enter customer password: qwertyuiop
Enter Customer's Address : oiuytre
Customer created successfully.
Customer registered successfully.
```

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.**

```
import mysql.connector
from mysql.connector import Error
from myexception import CustomerNotFoundException, ProductNotFoundException
class OrderProcessorRepositoryImpl:
    def __init__(self):
        self.connection = self.connect_to_database()

    def connect_to_database(self):
        try:
            connection = mysql.connector.connect(
                host='localhost',
                database='ecom',
                user='root',
                password='Sneha@2705'
            )
            if connection.is_connected():
                print("Connected to database successfully.")
                return connection
        except Error as e:
            print(f"Error connecting to database: {e}")
    def customer_exist(self, customer_id: int) -> bool:
        try:
```

```

        cursor = self.connection.cursor()

        sql = "SELECT COUNT(*) FROM customers WHERE customer_id = %s"

        cursor.execute(sql, (customer_id,))

        count = cursor.fetchone()[0]

        return count > 0

    except Error as e:

        print(f"Error checking customer existence: {e}")

        return False

    finally:

        if cursor:

            cursor.close()

def product_exist(self, product_id: int) -> bool:

    try:

        cursor = self.connection.cursor()

        sql = "SELECT COUNT(*) FROM products WHERE product_id = %s"

        cursor.execute(sql, (product_id,))

        count = cursor.fetchone()[0]

        return count > 0

    except Error as e:

        print(f"Error checking customer existence: {e}")

        return False

    finally:

        if cursor:

            cursor.close()

def create_product(self, product_id, name, price, description, stock_quantity):

    try:

        cursor = self.connection.cursor()

        sql = "INSERT INTO products (product_id, name, price, description, stockQuantity)
VALUES (%s, %s, %s, %s, %s)"

        cursor.execute(sql, (product_id, name, price, description, stock_quantity))

        self.connection.commit()

```

```

        print("Product created successfully.")
        return True
except Error as e:
    print(f'Error creating product: {e}')
    return False
finally:
    if cursor:
        cursor.close()
def addToCart(self, customer_id: int, product_id: int, quantity: int) -> bool:
    try:
        if not self.customer_exist(customer_id):
            raise CustomerNotFoundException(f'Customer with ID {customer_id} not found.')
        elif not self.product_exist(product_id):
            raise ProductNotFoundException(f'Product with ID {product_id} not found.')
        with self.connection.cursor() as cursor:
            sql = "INSERT INTO cart (customer_id, product_id, quantity) VALUES (%s, %s,
%s)"
            cursor.execute(sql, (customer_id, product_id, quantity))
        self.connection.commit()
        print("Product added to cart successfully.")
        return True
    except Error as e:
        print(f'Error adding product to cart: {e}')
        return False
def orderProduct(self, cart_id: int) -> bool:
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM cart WHERE cart_id = %s"
        cursor.execute(sql, (cart_id,))
        result = cursor.fetchall()
        if result:

```

```

        print("Product ordered successfully.")
        return True
    else:
        print("Order not found in cart.")
        return False
except Error as e:
    print(f"Error ordering product: {e}")
    return False
finally:
    if cursor:
        cursor.close()
def __del__(self):
    if self.connection:
        self.connection.close()
        print("Database connection closed.")

import unittest
from testing_dummy import OrderProcessorRepositoryImpl
from myexception import CustomerNotFoundException, ProductNotFoundException

class TestMain(unittest.TestCase):

    def setUp(self):
        self.tst = OrderProcessorRepositoryImpl()

    def test_product_created_successfully(self):
        product_id = self.tst.create_product(12, "laptop", 800.00, "High-performance laptop", 10)
        self.assertIsNotNone(product_id)
    def test_product_added_to_cart_successfully(self):
        result = self.tst.addToCart(1, 12, 1)

```



```
        self.assertTrue(result)

    def test_product_ordered_successfully(self):
        result = self.tst.orderProduct(2)

        self.assertTrue(result)

    def test_customer_not_found_exception(self):
        with self.assertRaises(CustomerNotFoundException):
            self.tst.addToCart(100, 12, 1)

    def test_product_not_found_exception(self):
        with self.assertRaises(ProductNotFoundException):
            self.tst.addToCart(1, 999, 1)

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

```
Connected to database successfully.
.Database connection closed.
Connected to database successfully.
Product added to cart successfully.
.Database connection closed.
Connected to database successfully.
Error creating product: 1062 (23000): Duplicate entry '12' for key 'products.PRIMARY'
.Database connection closed.
Connected to database successfully.
.Database connection closed.
Connected to database successfully.
Product ordered successfully.
.Database connection closed.

-----
Ran 5 tests in 0.454s

OK
```