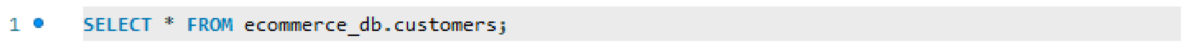
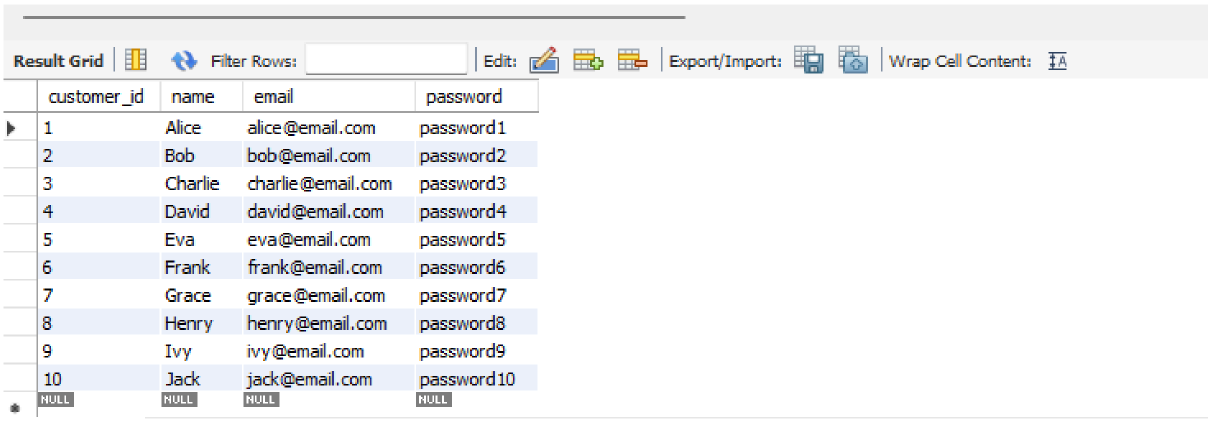
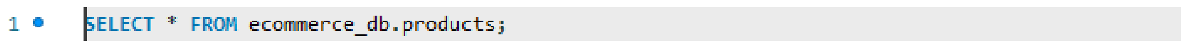
Create following tables in SQL Schema with appropriate class and write the unit test case for the
Ecommerce application.
Schema Design:
1. **customers** table:
• customer_id (Primary Key)
• name
• email
• password
2. **products** table:
• product_id (Primary Key)
• name
• price
• description
• stockQuantity
3. **cart** table:
• cart_id (Primary Key)
• customer_id (Foreign Key)
• product_id (Foreign Key)
• quantity
4. **orders** table:
• order_id (Primary Key)
• customer_id (Foreign Key)
• order_date
• total_price
• shipping_address
5. **order_items** table (to store order details):
• order_item_id (Primary Key)
• order_id (Foreign Key)
• product_id (Foreign Key)
• quantity



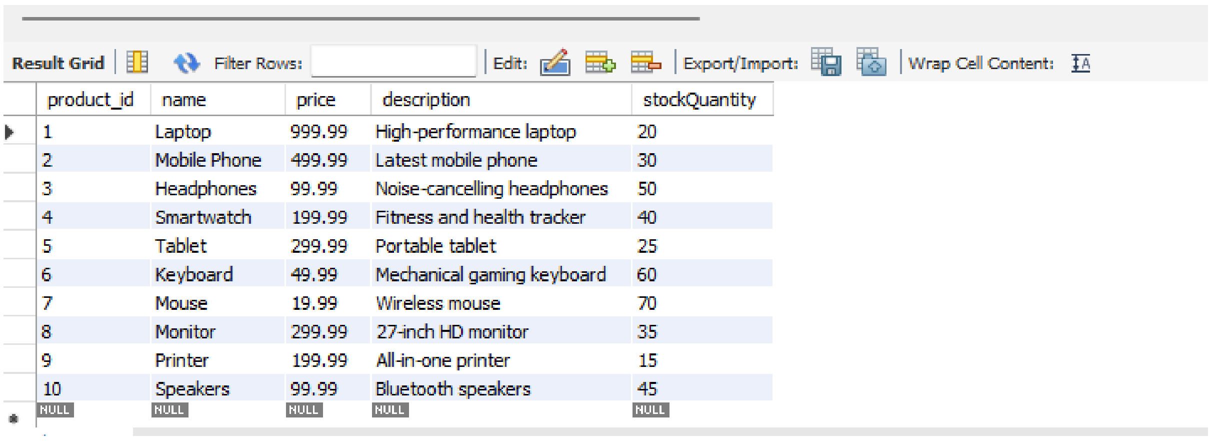
```sql
SELECT * FROM ecommerce_db.customers;
```



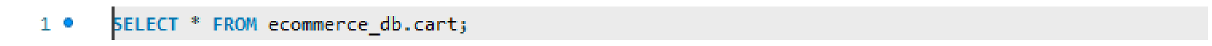
customer_id	name	email	password
1	Alice	alice@email.com	password1
2	Bob	bob@email.com	password2
3	Charlie	charlie@email.com	password3
4	David	david@email.com	password4
5	Eva	eva@email.com	password5
6	Frank	frank@email.com	password6
7	Grace	grace@email.com	password7
8	Henry	henry@email.com	password8
9	Ivy	ivy@email.com	password9
10	Jack	jack@email.com	password10
NULL	NULL	NULL	NULL



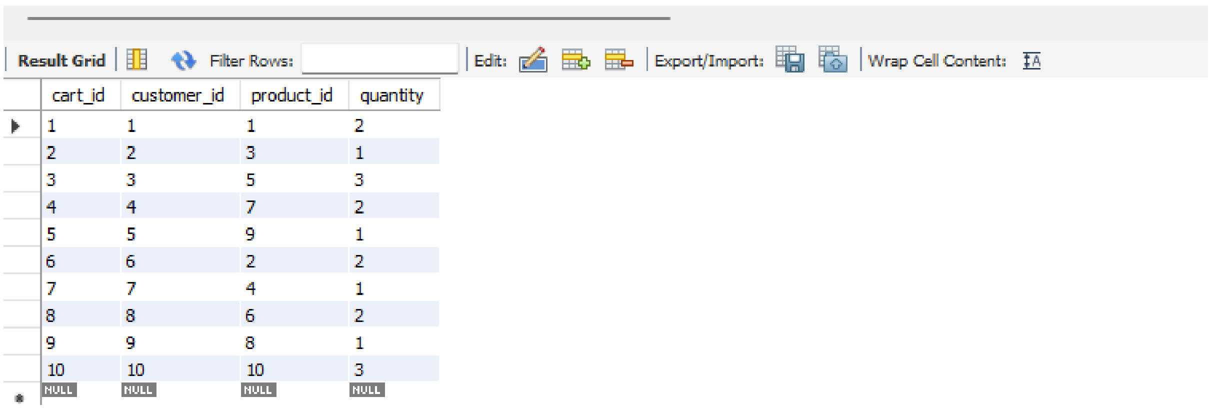
```sql
SELECT * FROM ecommerce_db.products;
```



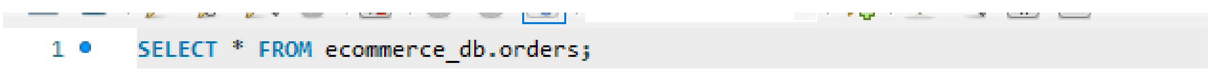
product_id	name	price	description	stockQuantity
1	Laptop	999.99	High-performance laptop	20
2	Mobile Phone	499.99	Latest mobile phone	30
3	Headphones	99.99	Noise-cancelling headphones	50
4	Smartwatch	199.99	Fitness and health tracker	40
5	Tablet	299.99	Portable tablet	25
6	Keyboard	49.99	Mechanical gaming keyboard	60
7	Mouse	19.99	Wireless mouse	70
8	Monitor	299.99	27-inch HD monitor	35
9	Printer	199.99	All-in-one printer	15
10	Speakers	99.99	Bluetooth speakers	45
NULL	NULL	NULL	NULL	NULL



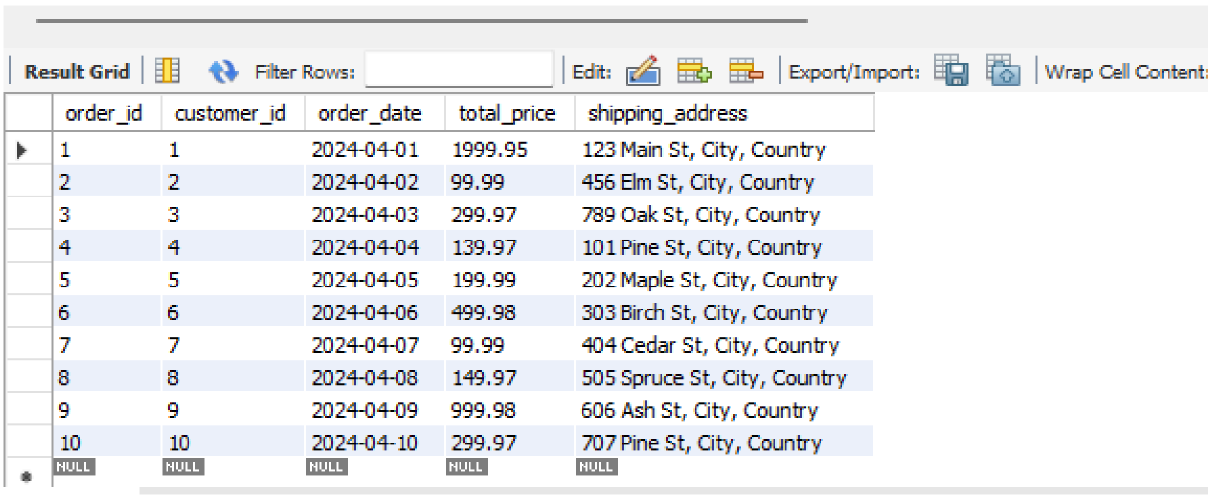
```sql
SELECT * FROM ecommerce_db.cart;
```



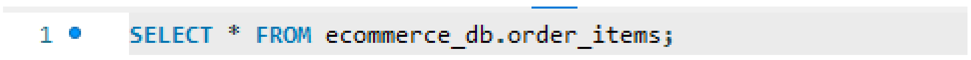
cart_id	customer_id	product_id	quantity
1	1	1	2
2	2	3	1
3	3	5	3
4	4	7	2
5	5	9	1
6	6	2	2
7	7	4	1
8	8	6	2
9	9	8	1
10	10	10	3
NULL	NULL	NULL	NULL



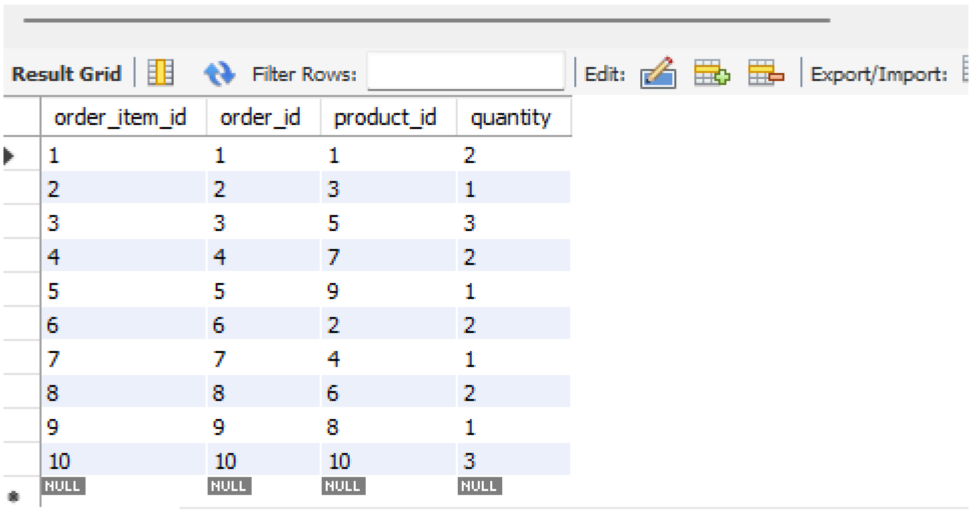
```sql
SELECT * FROM ecommerce_db.orders;
```



order_id	customer_id	order_date	total_price	shipping_address
1	1	2024-04-01	1999.95	123 Main St, City, Country
2	2	2024-04-02	99.99	456 Elm St, City, Country
3	3	2024-04-03	299.97	789 Oak St, City, Country
4	4	2024-04-04	139.97	101 Pine St, City, Country
5	5	2024-04-05	199.99	202 Maple St, City, Country
6	6	2024-04-06	499.98	303 Birch St, City, Country
7	7	2024-04-07	99.99	404 Cedar St, City, Country
8	8	2024-04-08	149.97	505 Spruce St, City, Country
9	9	2024-04-09	999.98	606 Ash St, City, Country
10	10	2024-04-10	299.97	707 Pine St, City, Country
NULL	NULL	NULL	NULL	NULL



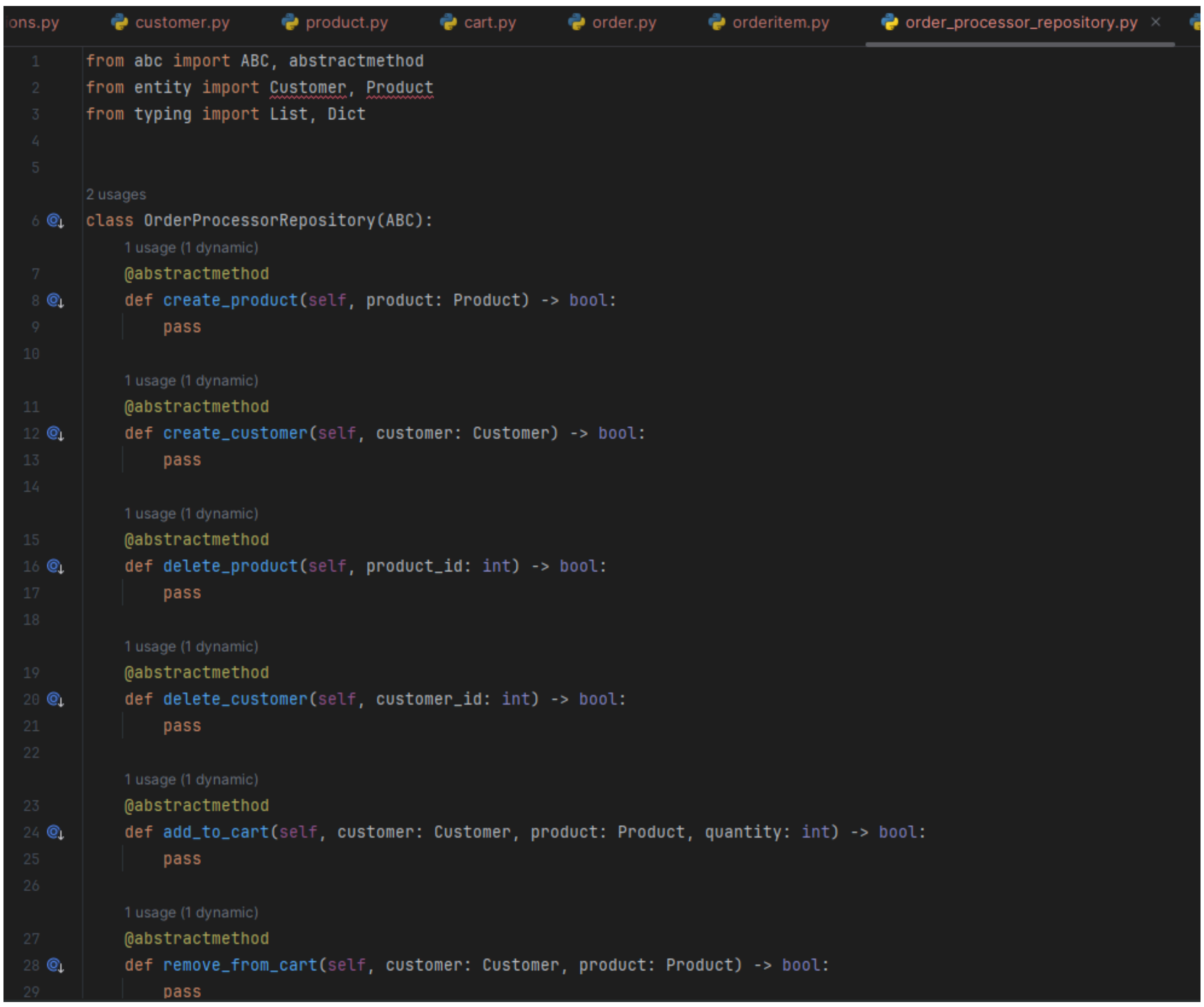
```
SELECT * FROM ecommerce_db.order_items;
```



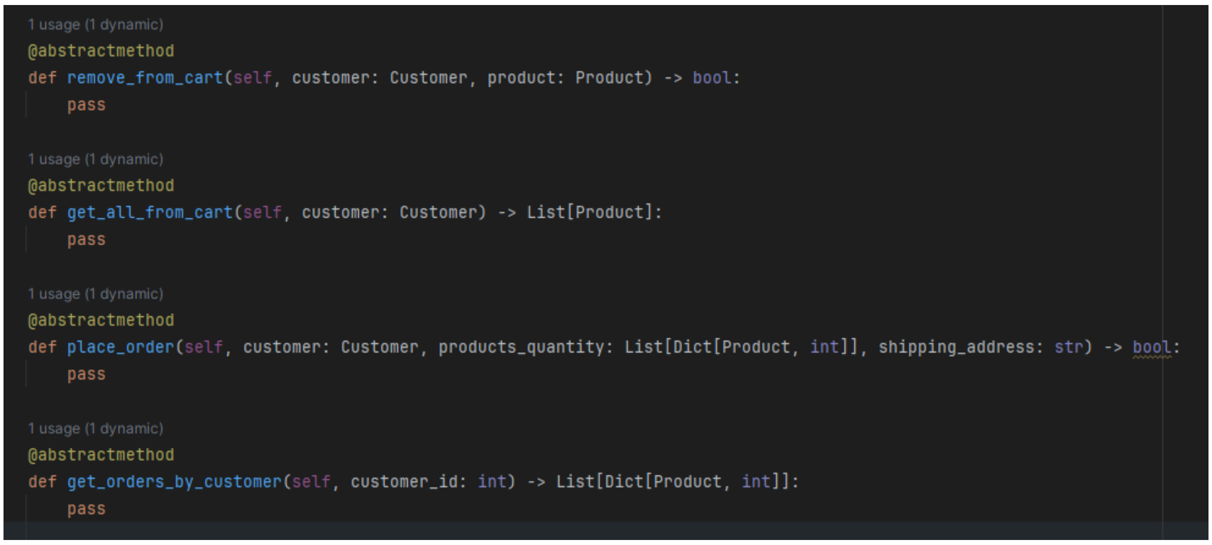
order_item_id	order_id	product_id	quantity
1	1	1	2
2	2	3	1
3	3	5	3
4	4	7	2
5	5	9	1
6	6	2	2
7	7	4	1
8	8	6	2
9	9	8	1
10	10	10	3
NULL	NULL	NULL	NULL

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© Hexaware Technologies Limited. All rights
www.hexaware.com
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

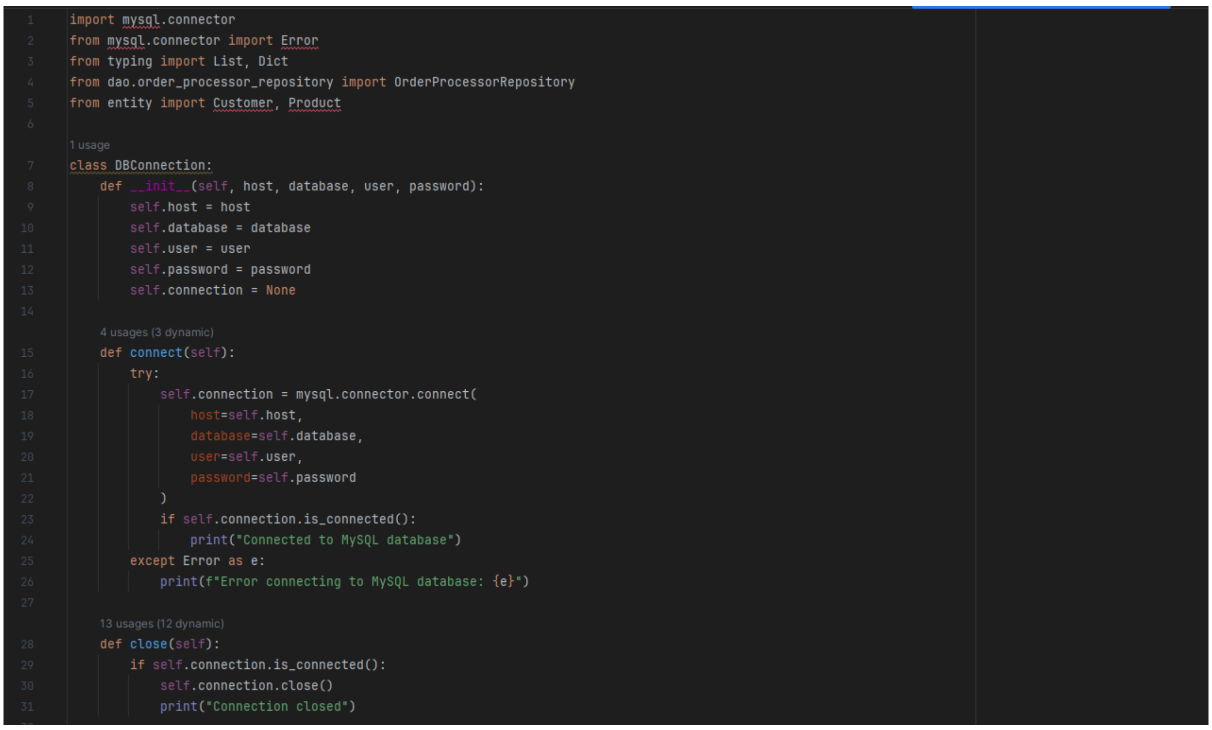


```python
from abc import ABC, abstractmethod
from entity import Customer, Product
from typing import List, Dict


2 usages
class OrderProcessorRepository(ABC):
    1 usage (1 dynamic)
    @abstractmethod
    def create_product(self, product: Product) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def create_customer(self, customer: Customer) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def delete_product(self, product_id: int) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def delete_customer(self, customer_id: int) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def remove_from_cart(self, customer: Customer, product: Product) -> bool:
        pass
```

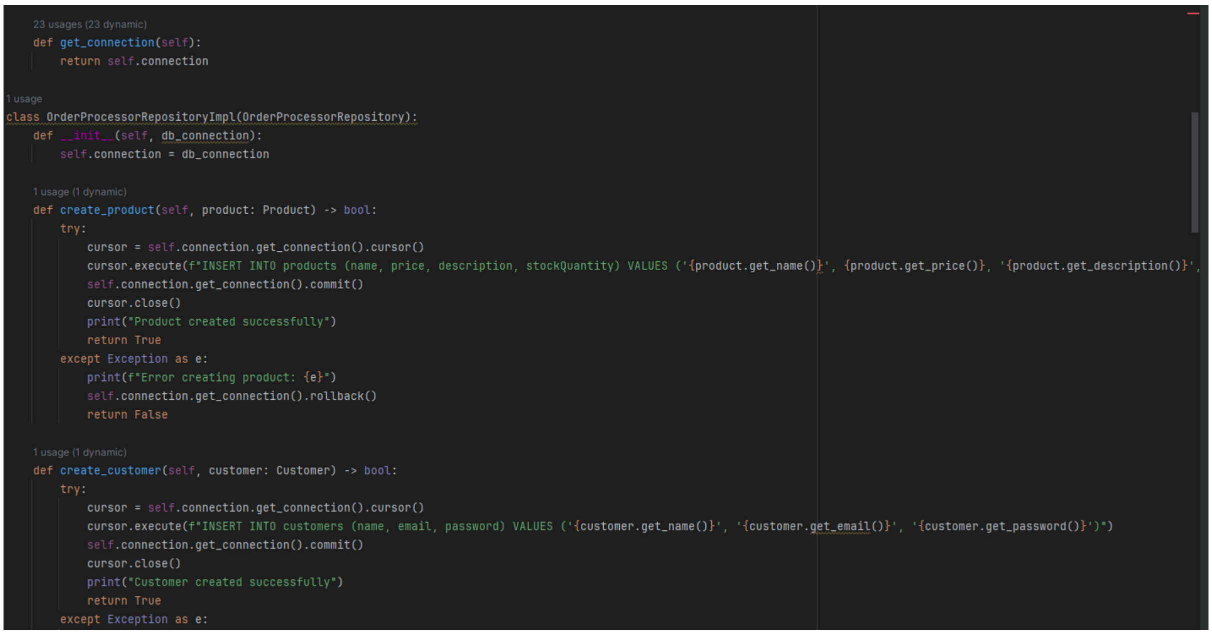


```
1 usage (1 dynamic)
@abstractmethod
def remove_from_cart(self, customer: Customer, product: Product) -> bool:
    pass

1 usage (1 dynamic)
@abstractmethod
def get_all_from_cart(self, customer: Customer) -> List[Product]:
    pass

1 usage (1 dynamic)
@abstractmethod
def place_order(self, customer: Customer, products_quantity: List[Dict[Product, int]], shipping_address: str) -> bool:
    pass

1 usage (1 dynamic)
@abstractmethod
def get_orders_by_customer(self, customer_id: int) -> List[Dict[Product, int]]:
    pass
```

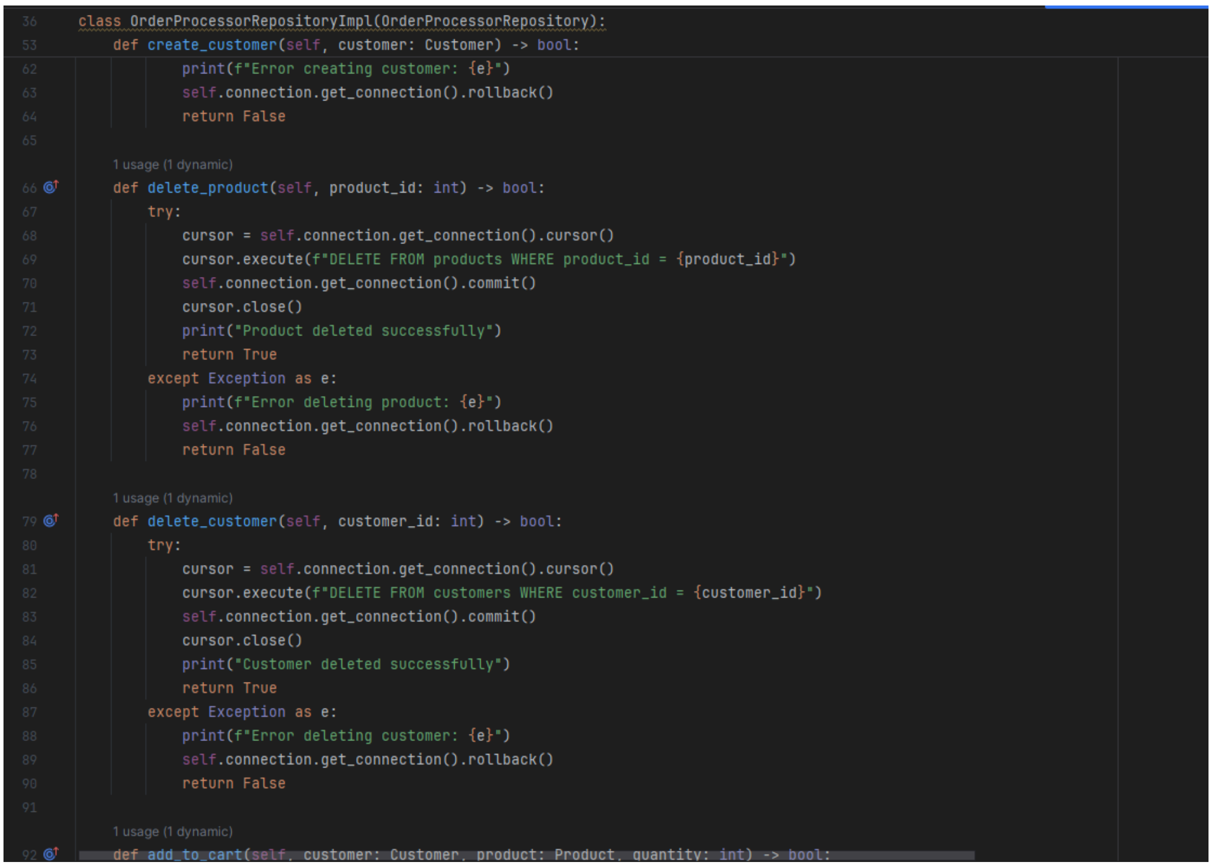
7. Implement the above interface in a class called **OrderProcessorRepositoryImpl in package dao**.



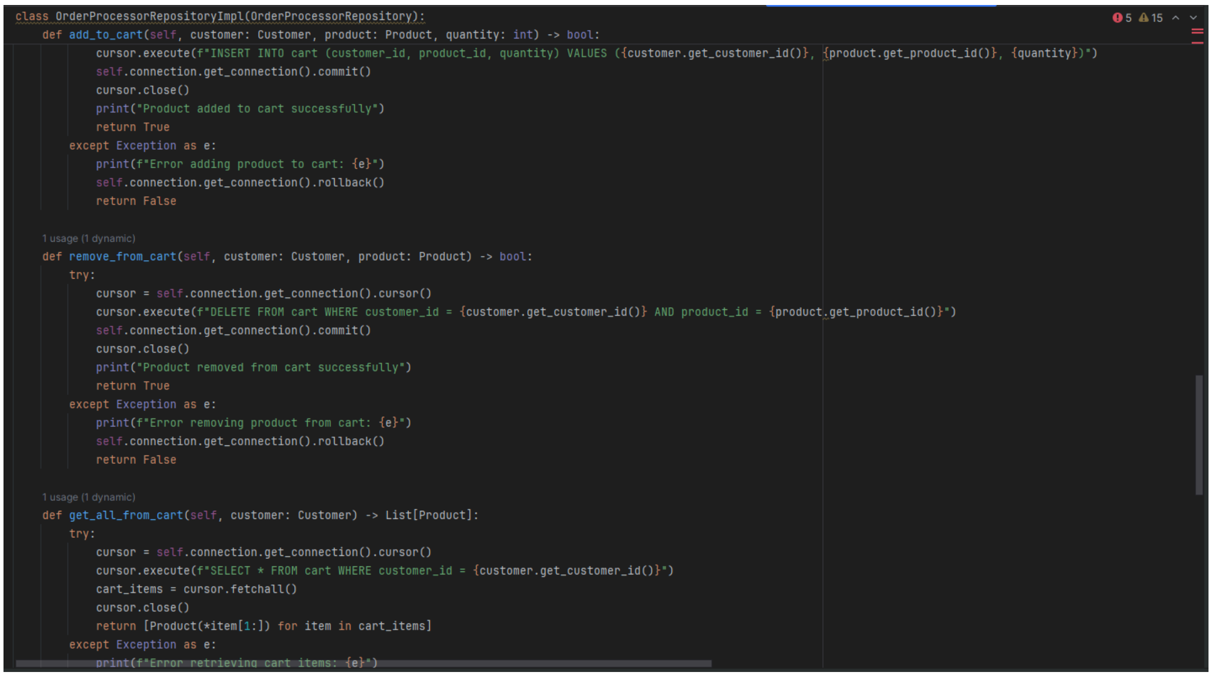
```
import mysql.connector
from mysql.connector import Error
from typing import List, Dict
from dao.order_processor_repository import OrderProcessorRepository
from entity import Customer, Product

1 usage
class DBConnection:
    def __init__(self, host, database, user, password):
        self.host = host
        self.database = database
        self.user = user
        self.password = password
        self.connection = None

    4 usages (3 dynamic)
    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                database=self.database,
                user=self.user,
                password=self.password
            )
            if self.connection.is_connected():
                print("Connected to MySQL database")
        except Error as e:
            print(f"Error connecting to MySQL database: {e}")

    13 usages (12 dynamic)
    def close(self):
        if self.connection.is_connected():
            self.connection.close()
            print("Connection closed")
```



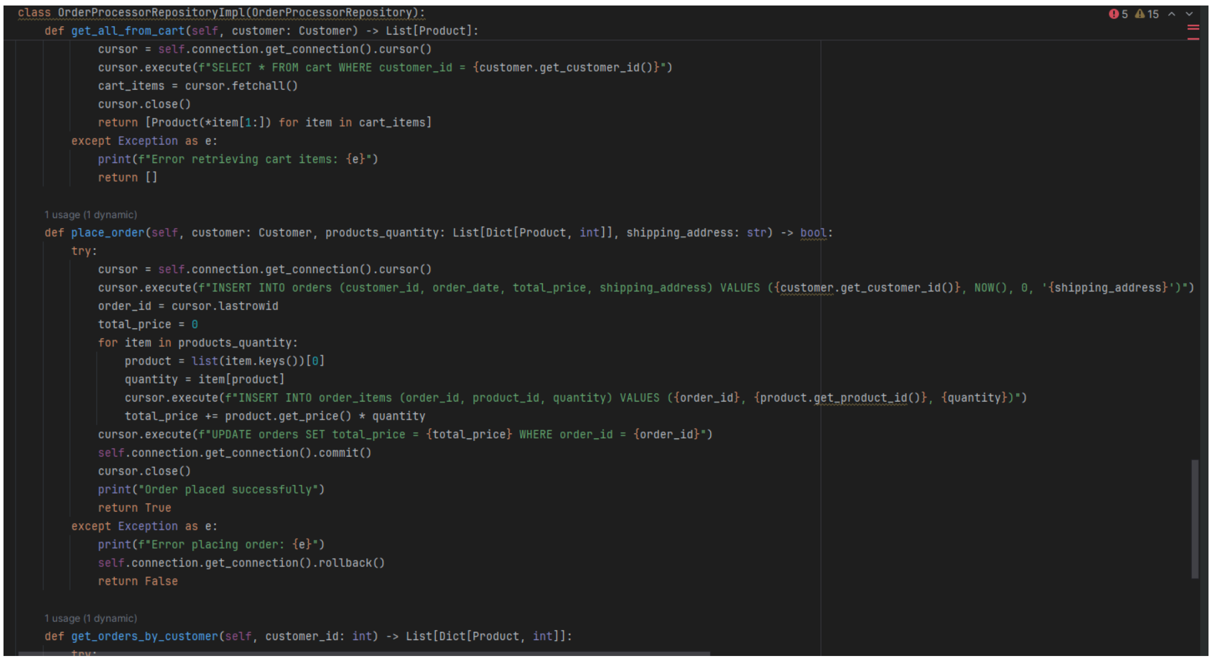
```
    23 usages (23 dynamic)
    def get_connection(self):
        return self.connection

1 usage
class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    def __init__(self, db_connection):
        self.connection = db_connection

    1 usage (1 dynamic)
    def create_product(self, product: Product) -> bool:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"INSERT INTO products (name, price, description, stockQuantity) VALUES ('{product.get_name()}', {product.get_price()}, '{product.get_description()}',
            self.connection.get_connection().commit()
            cursor.close()
            print("Product created successfully")
            return True
        except Exception as e:
            print(f"Error creating product: {e}")
            self.connection.get_connection().rollback()
            return False

    1 usage (1 dynamic)
    def create_customer(self, customer: Customer) -> bool:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"INSERT INTO customers (name, email, password) VALUES ('{customer.get_name()}', '{customer.get_email()}', '{customer.get_password()}')")
            self.connection.get_connection().commit()
            cursor.close()
            print("Customer created successfully")
            return True
        except Exception as e:
```



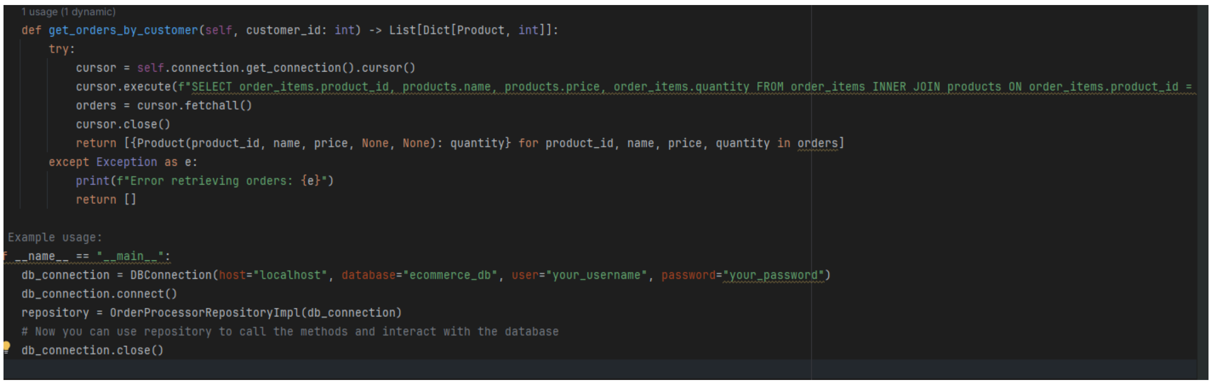
```
class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    def create_customer(self, customer: Customer) -> bool:
            print(f"Error creating customer: {e}")
            self.connection.get_connection().rollback()
            return False

    1 usage (1 dynamic)
    def delete_product(self, product_id: int) -> bool:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"DELETE FROM products WHERE product_id = {product_id}")
            self.connection.get_connection().commit()
            cursor.close()
            print("Product deleted successfully")
            return True
        except Exception as e:
            print(f"Error deleting product: {e}")
            self.connection.get_connection().rollback()
            return False

    1 usage (1 dynamic)
    def delete_customer(self, customer_id: int) -> bool:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"DELETE FROM customers WHERE customer_id = {customer_id}")
            self.connection.get_connection().commit()
            cursor.close()
            print("Customer deleted successfully")
            return True
        except Exception as e:
            print(f"Error deleting customer: {e}")
            self.connection.get_connection().rollback()
            return False

    1 usage (1 dynamic)
    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
```



```
class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    def add_to_cart(self, customer: Customer, product: Product, quantity: int) -> bool:
                cursor.execute(f"INSERT INTO cart (customer_id, product_id, quantity) VALUES ({customer.get_customer_id()}, {product.get_product_id()}, {quantity})")
                self.connection.get_connection().commit()
                cursor.close()
                print("Product added to cart successfully")
                return True
            except Exception as e:
                print(f"Error adding product to cart: {e}")
                self.connection.get_connection().rollback()
                return False

    1 usage (1 dynamic)
    def remove_from_cart(self, customer: Customer, product: Product) -> bool:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"DELETE FROM cart WHERE customer_id = {customer.get_customer_id()} AND product_id = {product.get_product_id()}")
            self.connection.get_connection().commit()
            cursor.close()
            print("Product removed from cart successfully")
            return True
        except Exception as e:
            print(f"Error removing product from cart: {e}")
            self.connection.get_connection().rollback()
            return False

    1 usage (1 dynamic)
    def get_all_from_cart(self, customer: Customer) -> List[Product]:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"SELECT * FROM cart WHERE customer_id = {customer.get_customer_id()}")
            cart_items = cursor.fetchall()
            cursor.close()
            return [Product(*item[1:]) for item in cart_items]
        except Exception as e:
            print(f"Error retrieving cart items: {e}")
```

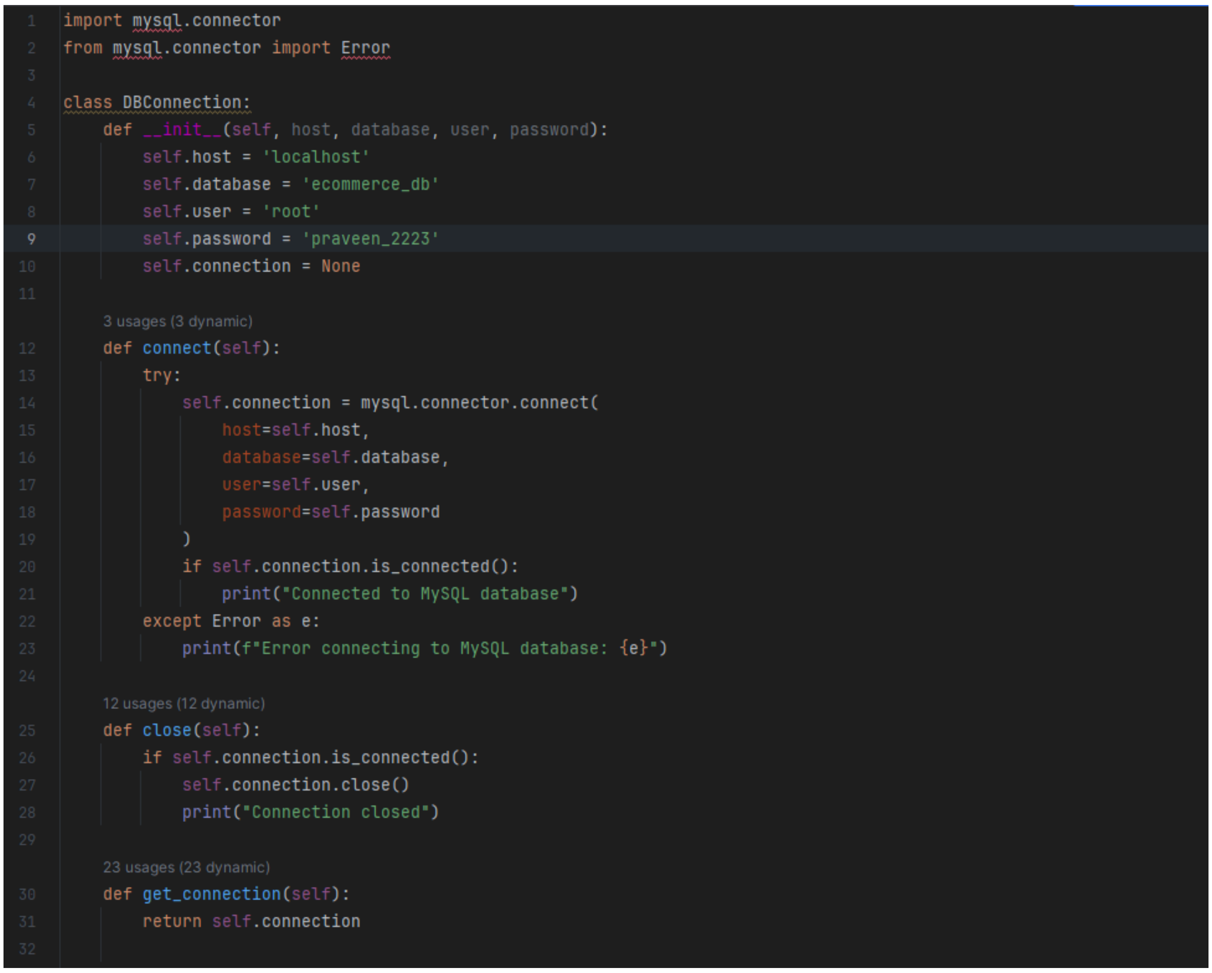


```
class OrderProcessorRepositoryImpl(OrderProcessorRepository):
    def get_all_from_cart(self, customer: Customer) -> List[Product]:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"SELECT * FROM cart WHERE customer_id = {customer.get_customer_id()}")
            cart_items = cursor.fetchall()
            cursor.close()
            return [Product(*item[1:]) for item in cart_items]
        except Exception as e:
            print(f"Error retrieving cart items: {e}")
            return []

    1 usage (1 dynamic)
    def place_order(self, customer: Customer, products_quantity: List[Dict[Product, int]], shipping_address: str) -> bool:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"INSERT INTO orders (customer_id, order_date, total_price, shipping_address) VALUES ({customer.get_customer_id()}, NOW(), 0, '{shipping_address}')")
            order_id = cursor.lastrowid
            total_price = 0
            for item in products_quantity:
                product = list(item.keys())[0]
                quantity = item[product]
                cursor.execute(f"INSERT INTO order_items (order_id, product_id, quantity) VALUES ({order_id}, {product.get_product_id()}, {quantity})")
                total_price += product.get_price() * quantity
            cursor.execute(f"UPDATE orders SET total_price = {total_price} WHERE order_id = {order_id}")
            self.connection.get_connection().commit()
            cursor.close()
            print("Order placed successfully")
            return True
        except Exception as e:
            print(f"Error placing order: {e}")
            self.connection.get_connection().rollback()
            return False

    1 usage (1 dynamic)
    def get_orders_by_customer(self, customer_id: int) -> List[Dict[Product, int]]:
        try:
```



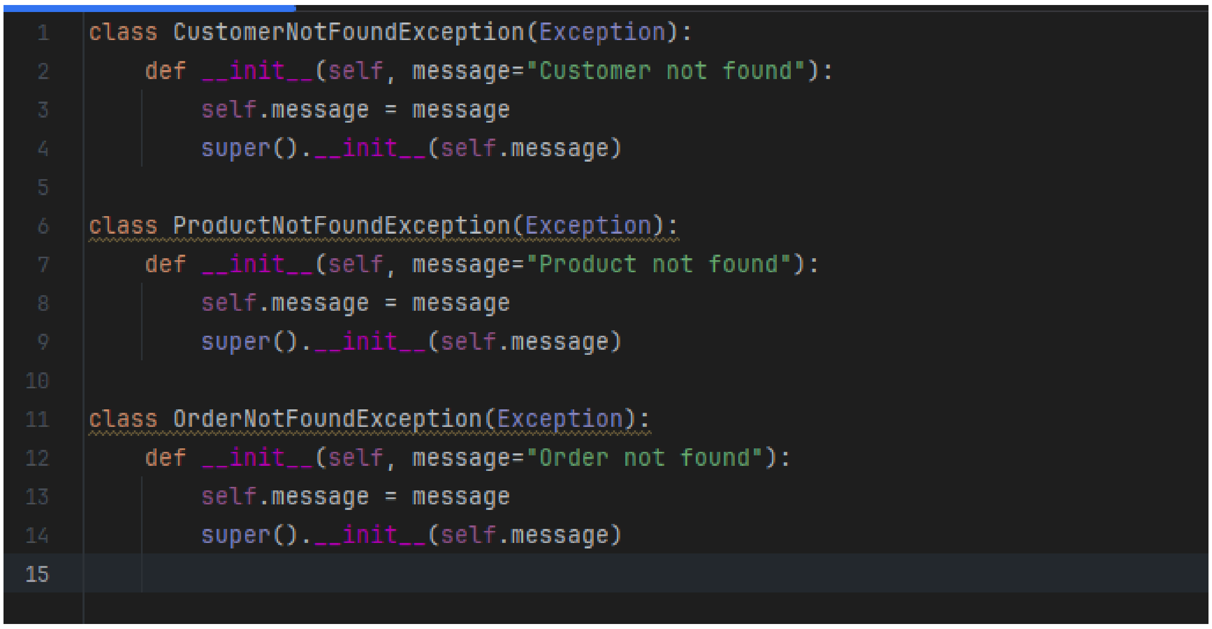
```
    1 usage (1 dynamic)
    def get_orders_by_customer(self, customer_id: int) -> List[Dict[Product, int]]:
        try:
            cursor = self.connection.get_connection().cursor()
            cursor.execute(f"SELECT order_items.product_id, products.name, products.price, order_items.quantity FROM order_items INNER JOIN products ON order_items.product_id =
            orders = cursor.fetchall()
            cursor.close()
            return [{Product(product_id, name, price, None, None): quantity} for product_id, name, price, quantity in orders]
        except Exception as e:
            print(f"Error retrieving orders: {e}")
            return []

Example usage:
f __name__ == "__main__":
    db_connection = DBConnection(host="localhost", database="ecommerce_db", user="your_username", password="your_password")
    db_connection.connect()
    repository = OrderProcessorRepositoryImpl(db_connection)
    # Now you can use repository to call the methods and interact with the database
    db_connection.close()
```

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



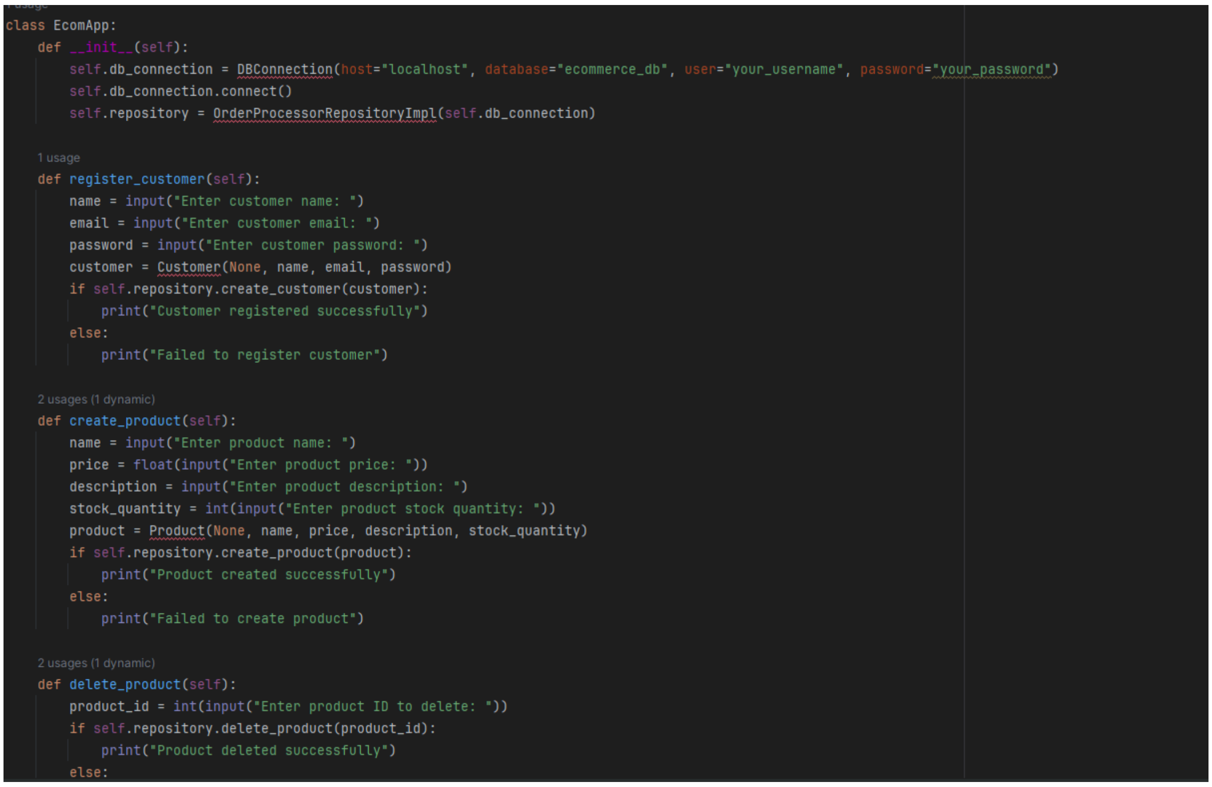
```
import mysql.connector
from mysql.connector import Error

class DBConnection:
    def __init__(self, host, database, user, password):
        self.host = 'localhost'
        self.database = 'ecommerce_db'
        self.user = 'root'
        self.password = 'praveen_2223'
        self.connection = None

    3 usages (3 dynamic)
    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                database=self.database,
                user=self.user,
                password=self.password
            )
            if self.connection.is_connected():
                print("Connected to MySQL database")
        except Error as e:
            print(f"Error connecting to MySQL database: {e}")

    12 usages (12 dynamic)
    def close(self):
        if self.connection.is_connected():
            self.connection.close()
            print("Connection closed")

    23 usages (23 dynamic)
    def get_connection(self):
        return self.connection

```

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

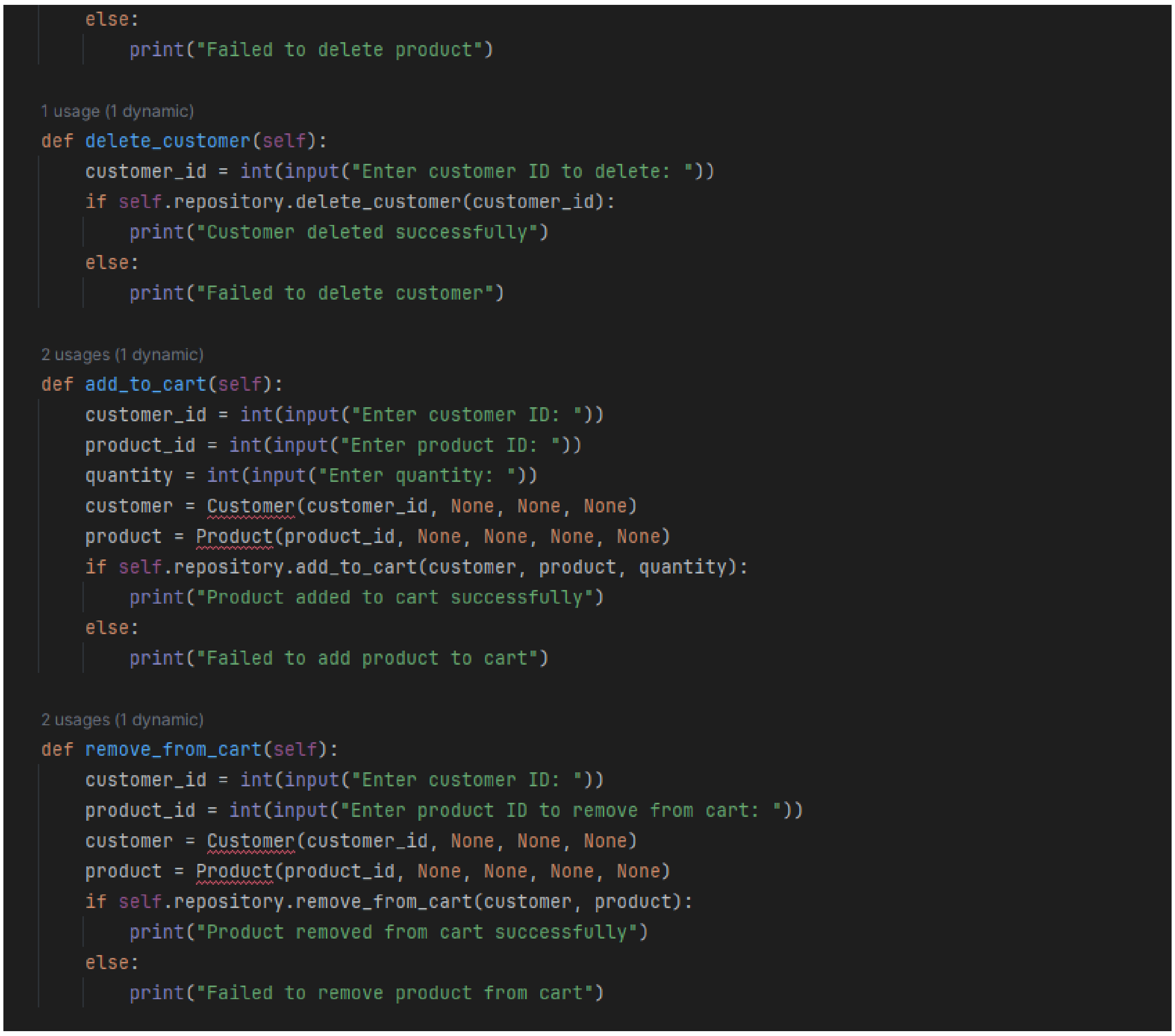


```python
class CustomerNotFoundException(Exception):
    def __init__(self, message="Customer not found"):
        self.message = message
        super().__init__(self.message)

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)

```

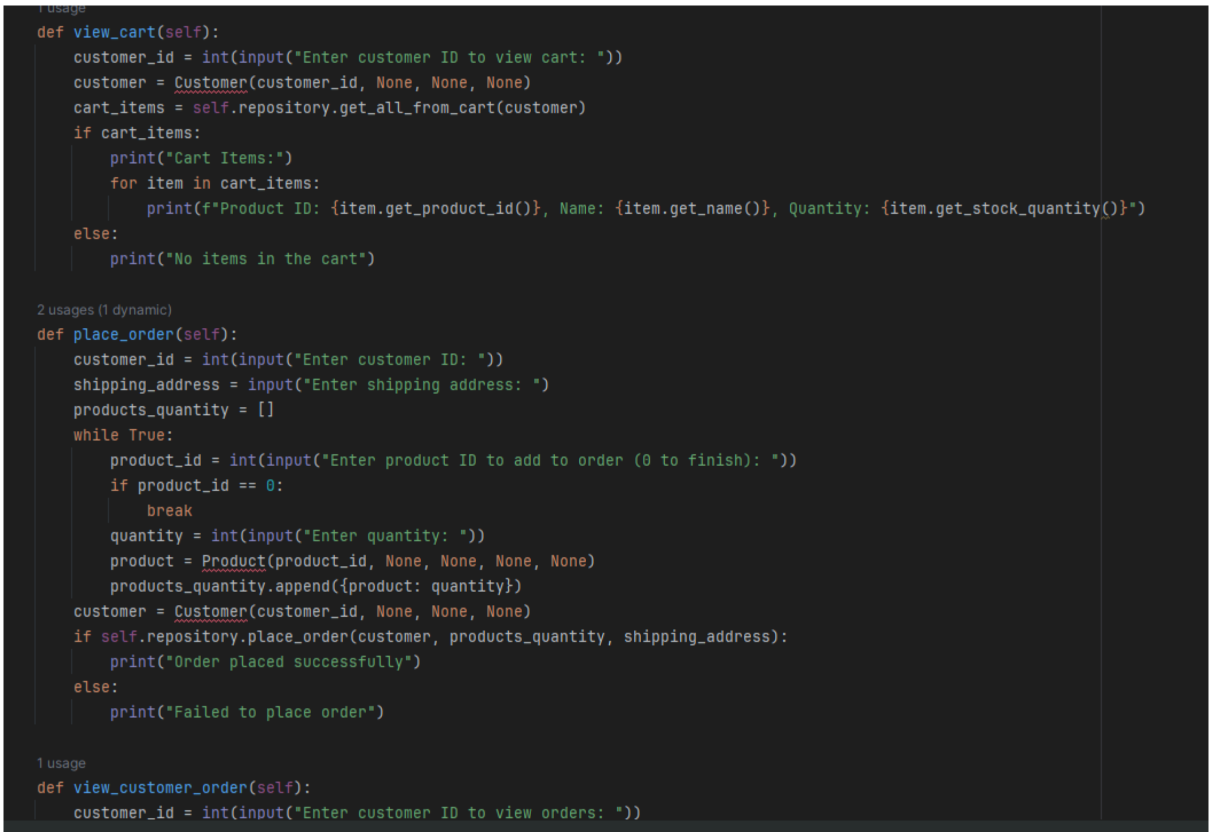
10. Create class named **EcomApp** with main method in app Trigger all the methods in service
implementation class by user choose operation from the following menu.
1. Register Customer.
2. Create Product.
3. Delete Product.
4. Add to cart.
5. View cart.
6. Place order.
7. View Customer Order



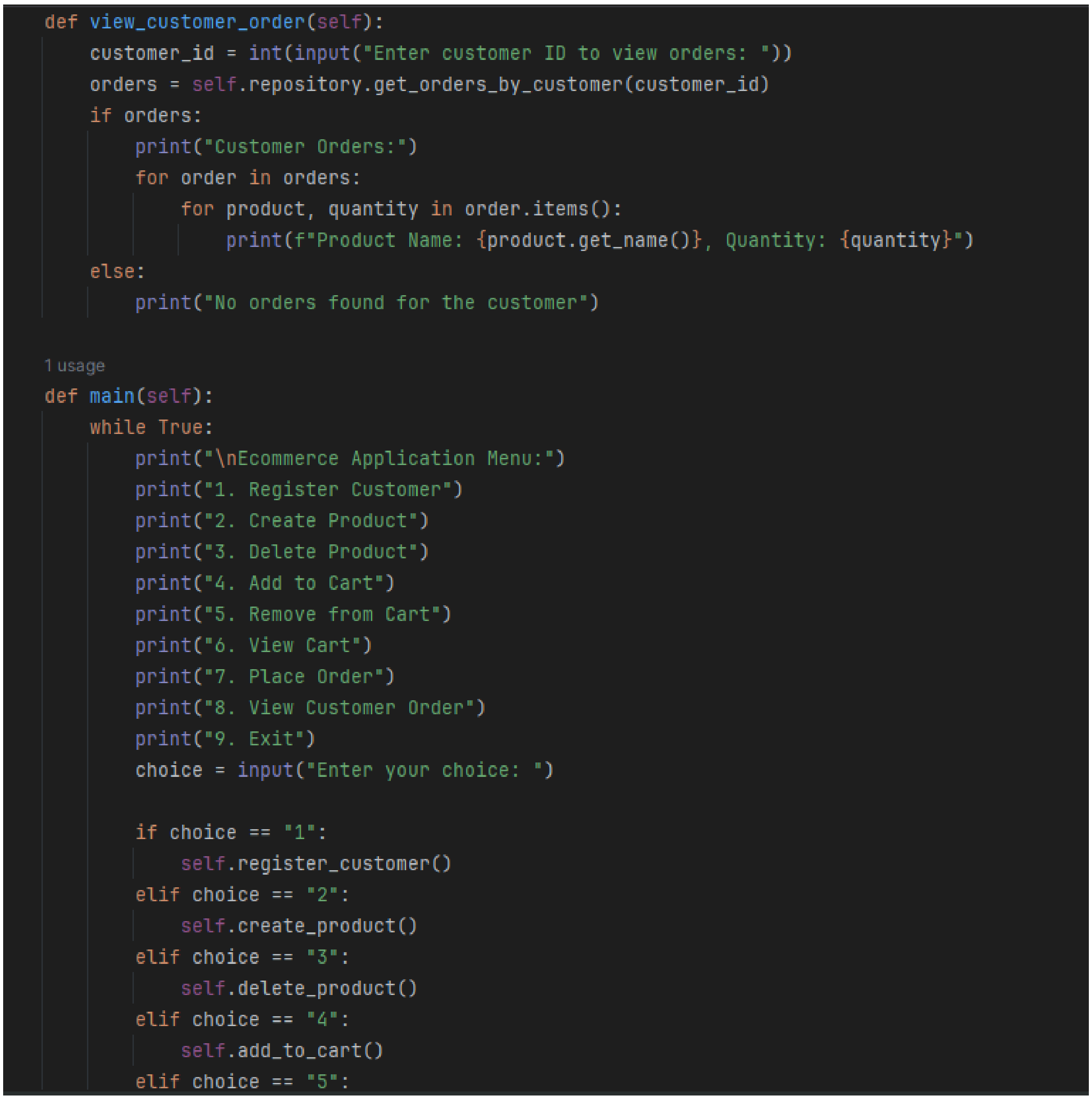
```python
class EcomApp:
    def __init__(self):
        self.db_connection = DBConnection(host="localhost", database="ecommerce_db", user="your_username", password="your_password")
        self.db_connection.connect()
        self.repository = OrderProcessorRepositoryImpl(self.db_connection)

    1 usage
    def register_customer(self):
        name = input("Enter customer name: ")
        email = input("Enter customer email: ")
        password = input("Enter customer password: ")
        customer = Customer(None, name, email, password)
        if self.repository.create_customer(customer):
            print("Customer registered successfully")
        else:
            print("Failed to register customer")

    2 usages (1 dynamic)
    def create_product(self):
        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(None, name, price, description, stock_quantity)
        if self.repository.create_product(product):
            print("Product created successfully")
        else:
            print("Failed to create product")

    2 usages (1 dynamic)
    def delete_product(self):
        product_id = int(input("Enter product ID to delete: "))
        if self.repository.delete_product(product_id):
            print("Product deleted successfully")
        else:
```



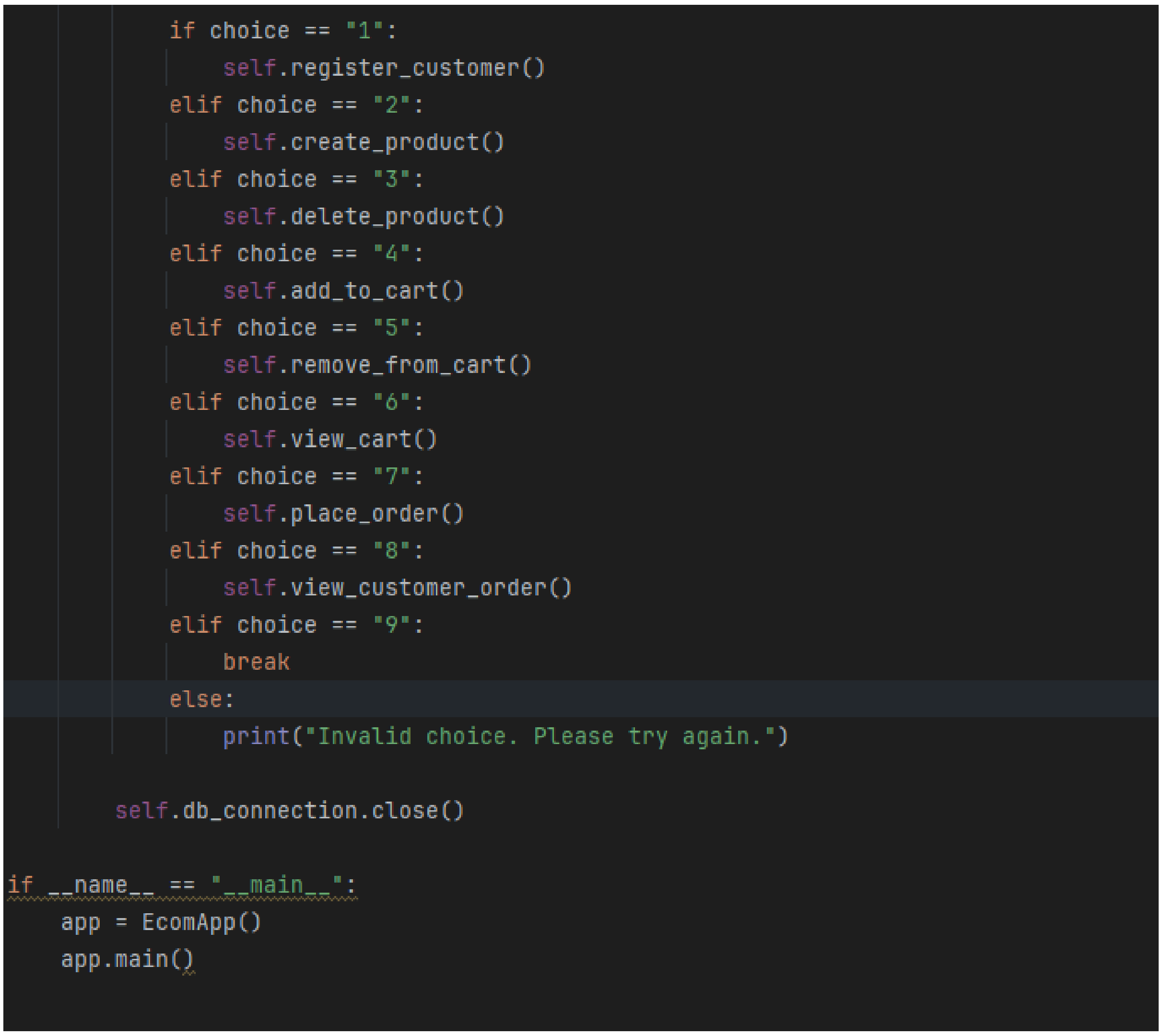
```
        else:
            print("Failed to delete product")

    1 usage (1 dynamic)
    def delete_customer(self):
        customer_id = int(input("Enter customer ID to delete: "))
        if self.repository.delete_customer(customer_id):
            print("Customer deleted successfully")
        else:
            print("Failed to delete customer")

    2 usages (1 dynamic)
    def add_to_cart(self):
        customer_id = int(input("Enter customer ID: "))
        product_id = int(input("Enter product ID: "))
        quantity = int(input("Enter quantity: "))
        customer = Customer(customer_id, None, None, None)
        product = Product(product_id, None, None, None, None)
        if self.repository.add_to_cart(customer, product, quantity):
            print("Product added to cart successfully")
        else:
            print("Failed to add product to cart")

    2 usages (1 dynamic)
    def remove_from_cart(self):
        customer_id = int(input("Enter customer ID: "))
        product_id = int(input("Enter product ID to remove from cart: "))
        customer = Customer(customer_id, None, None, None)
        product = Product(product_id, None, None, None, None)
        if self.repository.remove_from_cart(customer, product):
            print("Product removed from cart successfully")
        else:
            print("Failed to remove product from cart")
```



```
1 usage
def view_cart(self):
    customer_id = int(input("Enter customer ID to view cart: "))
    customer = Customer(customer_id, None, None, None)
    cart_items = self.repository.get_all_from_cart(customer)
    if cart_items:
        print("Cart Items:")
        for item in cart_items:
            print(f"Product ID: {item.get_product_id()}, Name: {item.get_name()}, Quantity: {item.get_stock_quantity()}")
    else:
        print("No items in the cart")

2 usages (1 dynamic)
def place_order(self):
    customer_id = int(input("Enter customer ID: "))
    shipping_address = input("Enter shipping address: ")
    products_quantity = []
    while True:
        product_id = int(input("Enter product ID to add to order (0 to finish): "))
        if product_id == 0:
            break
        quantity = int(input("Enter quantity: "))
        product = Product(product_id, None, None, None, None)
        products_quantity.append({product: quantity})
    customer = Customer(customer_id, None, None, None)
    if self.repository.place_order(customer, products_quantity, shipping_address):
        print("Order placed successfully")
    else:
        print("Failed to place order")

1 usage
def view_customer_order(self):
    customer_id = int(input("Enter customer ID to view orders: "))
```



```
def view_customer_order(self):
    customer_id = int(input("Enter customer ID to view orders: "))
    orders = self.repository.get_orders_by_customer(customer_id)
    if orders:
        print("Customer Orders:")
        for order in orders:
            for product, quantity in order.items():
                print(f"Product Name: {product.get_name()}, Quantity: {quantity}")
    else:
        print("No orders found for the customer")

1 usage
def main(self):
    while True:
        print("\nEcommerce Application Menu:")
        print("1. Register Customer")
        print("2. Create Product")
        print("3. Delete Product")
        print("4. Add to Cart")
        print("5. Remove from Cart")
        print("6. View Cart")
        print("7. Place Order")
        print("8. View Customer Order")
        print("9. 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":
```



```python
            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.remove_from_cart()
            elif choice == "6":
                self.view_cart()
            elif choice == "7":
                self.place_order()
            elif choice == "8":
                self.view_customer_order()
            elif choice == "9":
                break
            else:
                print("Invalid choice. Please try again.")

        self.db_connection.close()

if __name__ == "__main__":
    app = EcomApp()
    app.main()
```