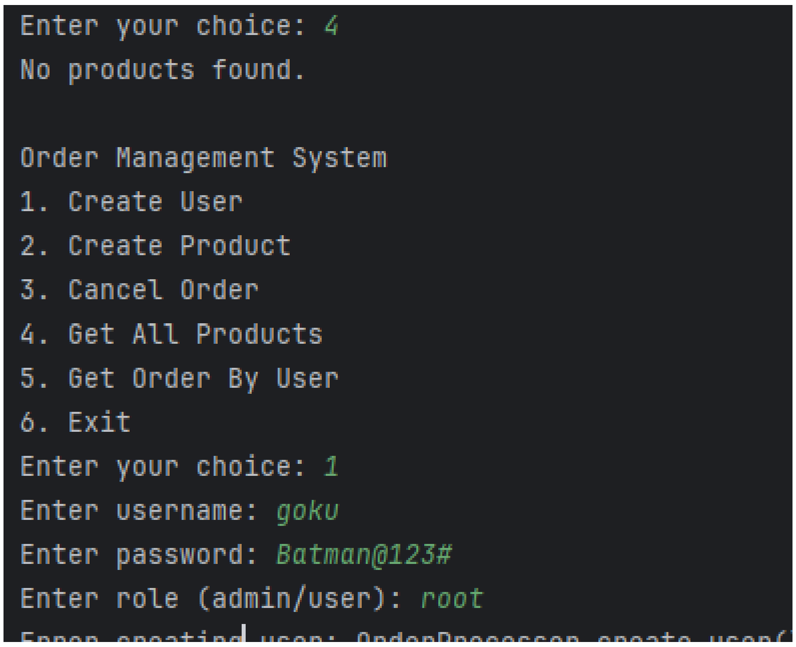
Order management System

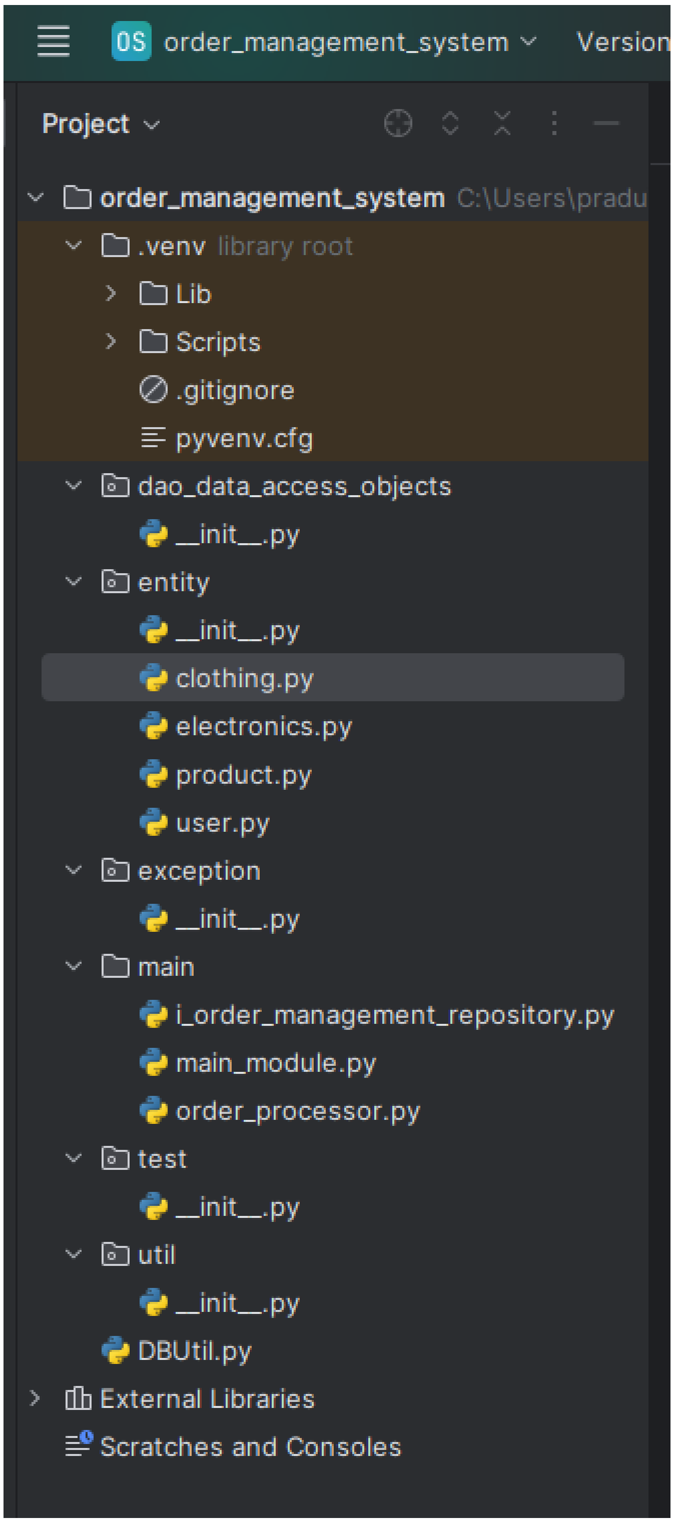
Python Assessment | Pradum singh

- ***Note : once you run the main module file something like this will appear on your terminal.from here on you can perform various operations in the database by applying various operations using interfaces/concrete class etc.***
- == || ==



```
Enter your choice: 4
No products found.

Order Management System
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Order By User
6. Exit
Enter your choice: 1
Enter username: goku
Enter password: Batman@123#
Enter role (admin/user): root
Error creating user: OrderProcessor.create_user(
```

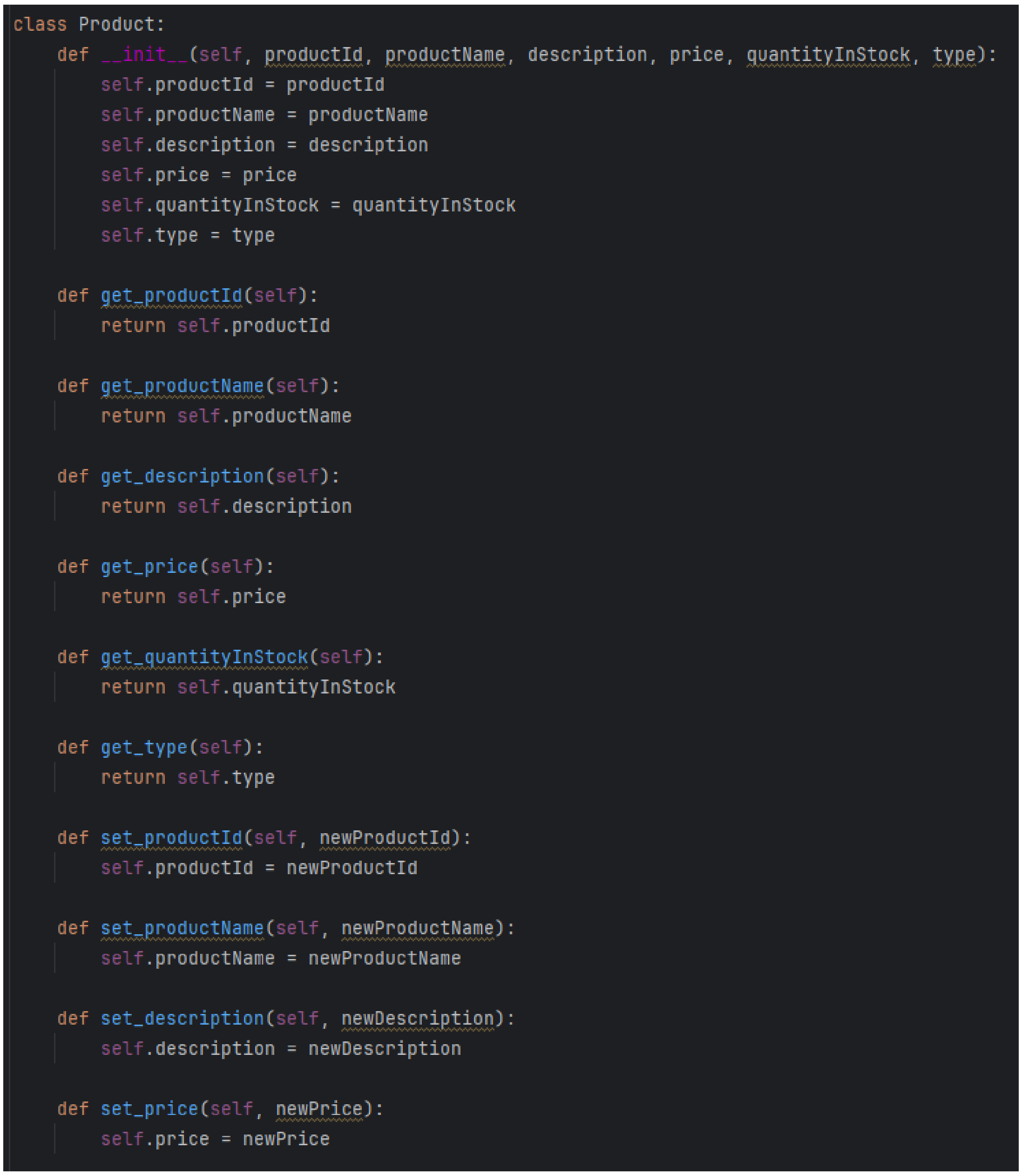
- ***Directory structure***


```
order_management_system    Version
Project
order_management_system C:\Users\pradu
    .venv library root
        Lib
        Scripts
        .gitignore
        pyvenv.cfg
    dao_data_access_objects
        __init__.py
    entity
        __init__.py
        clothing.py
        electronics.py
        product.py
        user.py
    exception
        __init__.py
    main
        i_order_management_repository.py
        main_module.py
        order_processor.py
    test
        __init__.py
    util
        __init__.py
    DBUtil.py
External Libraries
Scratches and Consoles
```

- ***TASK 1 - 5***
    - ***TASK 1 : Create a base class called Product with the following attributes:***

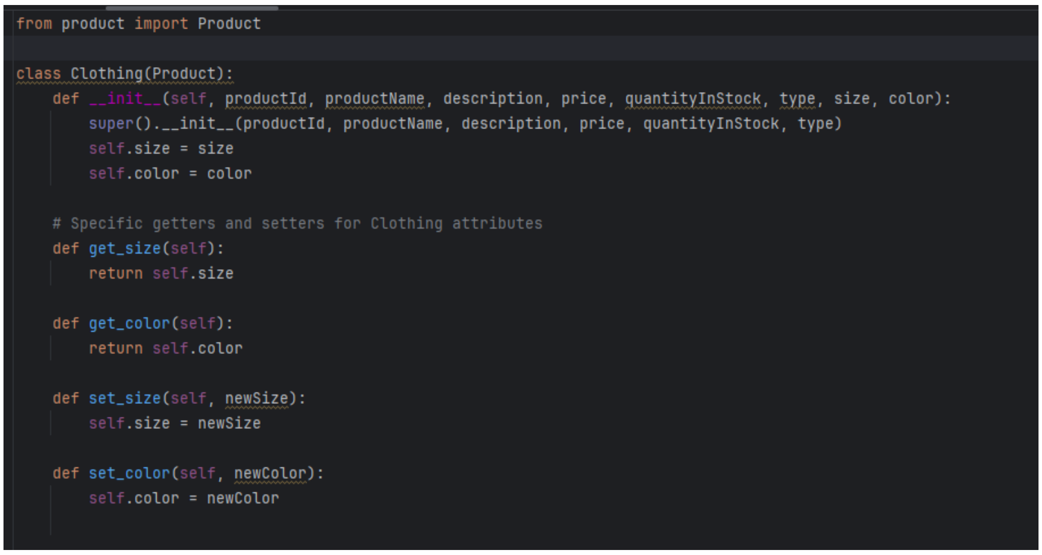
```
• productId (int)
• productName (String)
• description (String)
```

• *price (double)*
• *quantityInStock (int)*
• *type (String) [Electronics/Clothing]*



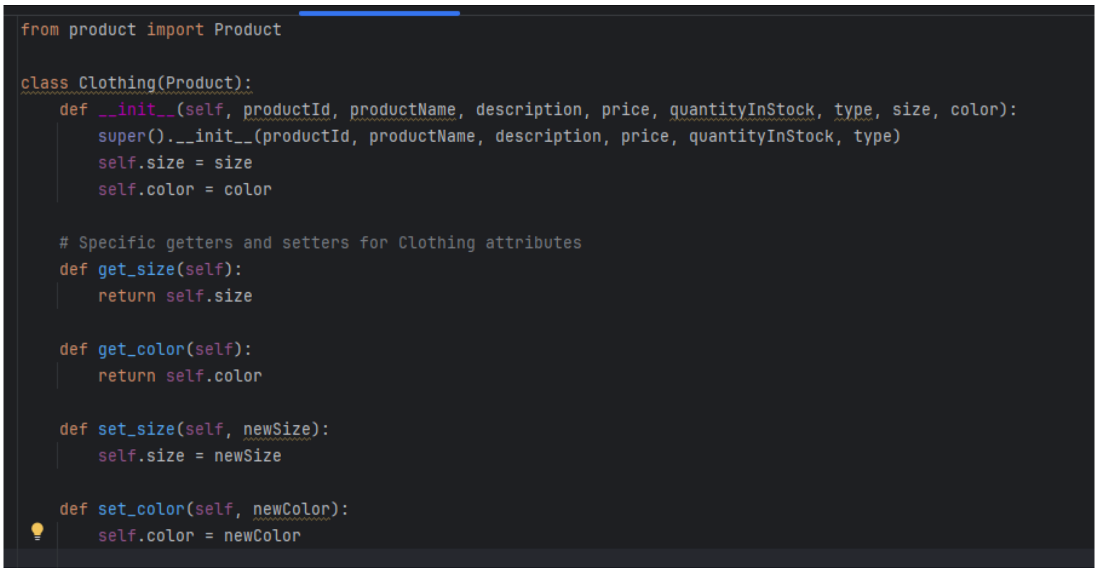
```python
class Product:
    def __init__(self, productId, productName, description, price, quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.type = type

    def get_productId(self):
        return self.productId

    def get_productName(self):
        return self.productName

    def get_description(self):
        return self.description

    def get_price(self):
        return self.price

    def get_quantityInStock(self):
        return self.quantityInStock

    def get_type(self):
        return self.type

    def set_productId(self, newProductId):
        self.productId = newProductId

    def set_productName(self, newProductName):
        self.productName = newProductName

    def set_description(self, newDescription):
        self.description = newDescription

    def set_price(self, newPrice):
        self.price = newPrice
```

- ***TASK 2 : Implement constructors, getters, and setters for the Product class. [implemented] check the code above and below***
- ***TASK 3 : Create a subclass Electronics that inherits from Product. Add attributes specific to electronicsproducts, such as:***
    - • *brand (String)*
      • *warrantyPeriod (int)*



```python
from product import Product

class Clothing(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.size = size
        self.color = color

    # Specific getters and setters for Clothing attributes
    def get_size(self):
        return self.size

    def get_color(self):
        return self.color

    def set_size(self, newSize):
        self.size = newSize

    def set_color(self, newColor):
        self.color = newColor
```

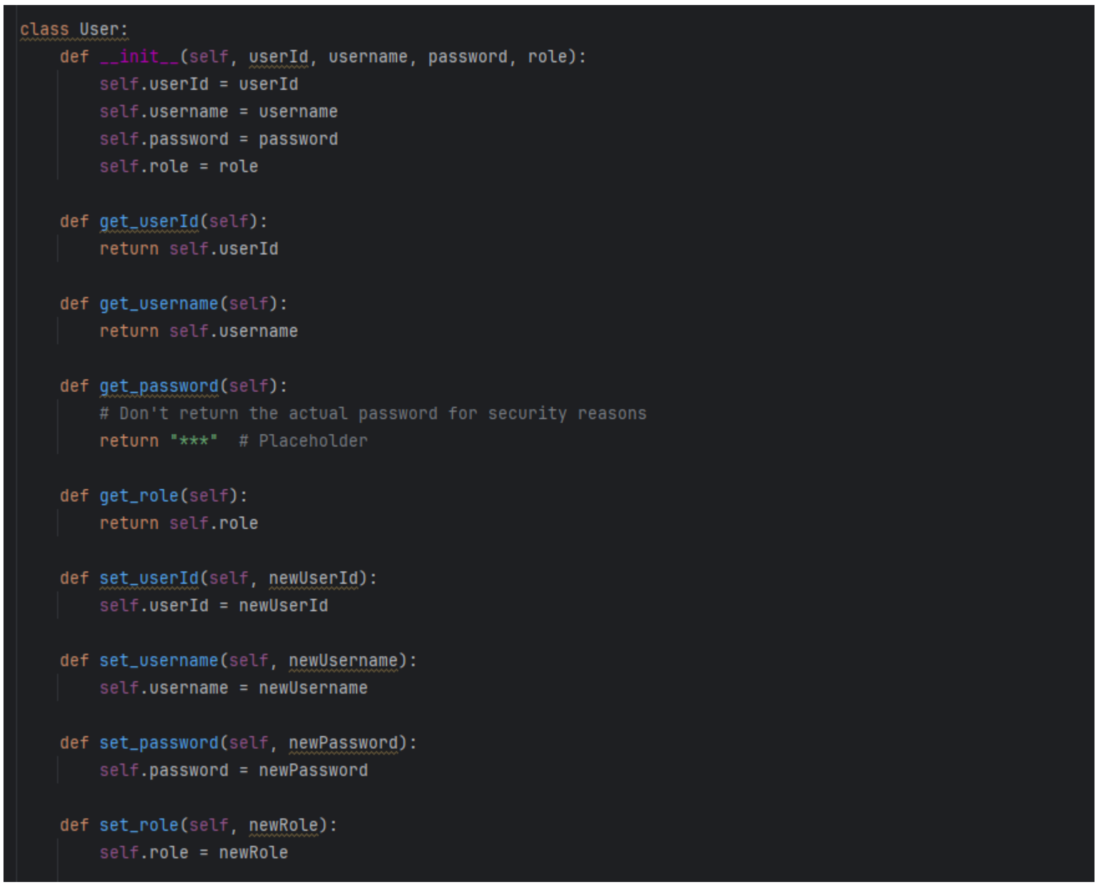
- ***TASK 4 :Create a subclass Clothing that also inherits from Product. Add attributes specific to clothing products, such as:***
  • size (String)
  • color (String)



```python
from product import Product

class Clothing(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.size = size
        self.color = color

    # Specific getters and setters for Clothing attributes
    def get_size(self):
        return self.size

    def get_color(self):
        return self.color

    def set_size(self, newSize):
        self.size = newSize

    def set_color(self, newColor):
        self.color = newColor
```

- ***TASK 5 : Create a User class with attributes:***

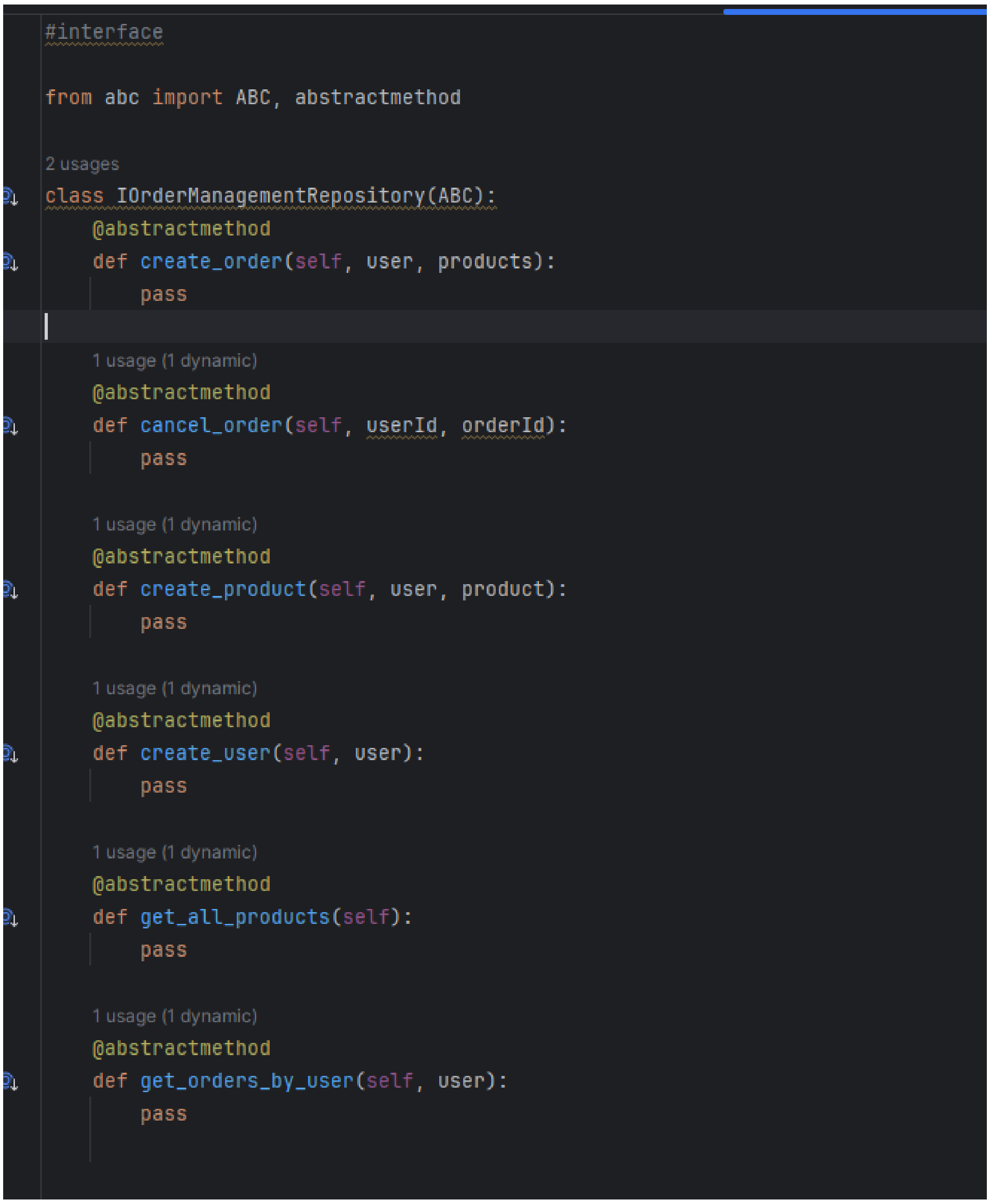
```
 • userId (int)
• username (String)
• password (String)
• role (String) // "Admin" or "User"
```



```python
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        self.role = role

    def get_userId(self):
        return self.userId

    def get_username(self):
        return self.username

    def get_password(self):
        # Don't return the actual password for security reasons
        return "***"  # Placeholder

    def get_role(self):
        return self.role

    def set_userId(self, newUserId):
        self.userId = newUserId

    def set_username(self, newUsername):
        self.username = newUsername

    def set_password(self, newPassword):
        self.password = newPassword

    def set_role(self, newRole):
        self.role = newRole
```

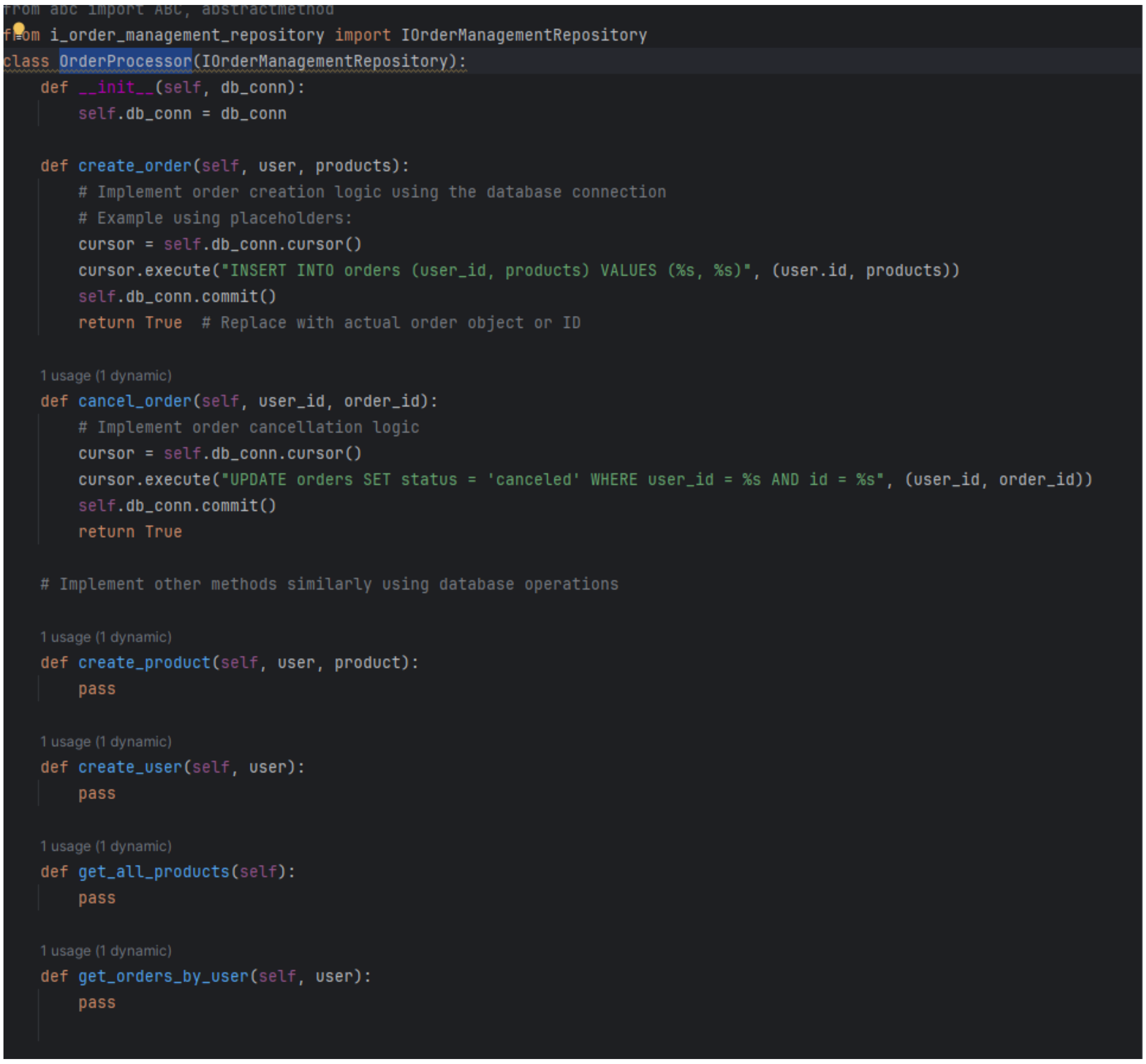
- ***TASK 6 : Define an interface/abstract class named IOrderManagementRepository with methods for:***
    - `createOrder(User user, list of products):` *check the user as already present in database to create order or create user (store in database) and create order.*
    - `cancelOrder(int userId, int orderId):` *check the userid and orderId already present in database and cancel the order. if any userId or orderId not present in database throw exception corresponding* `UserNotFound or OrderNotFound exception`

      • createProduct(User user, Product product): check the admin user as already present in database and create product and store in database.
    - `createUser(User user):` *create user and store in database for further development.*
    - `getAllProducts():` *return all product list from the database.*
    - `getOrderByUser(User user):` *return all product ordered by specific user from database.*



```
#interface

from abc import ABC, abstractmethod

2 usages
class IOrderManagementRepository(ABC):
    @abstractmethod
    def create_order(self, user, products):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def cancel_order(self, userId, orderId):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def create_product(self, user, product):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def create_user(self, user):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def get_all_products(self):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def get_orders_by_user(self, user):
        pass
```

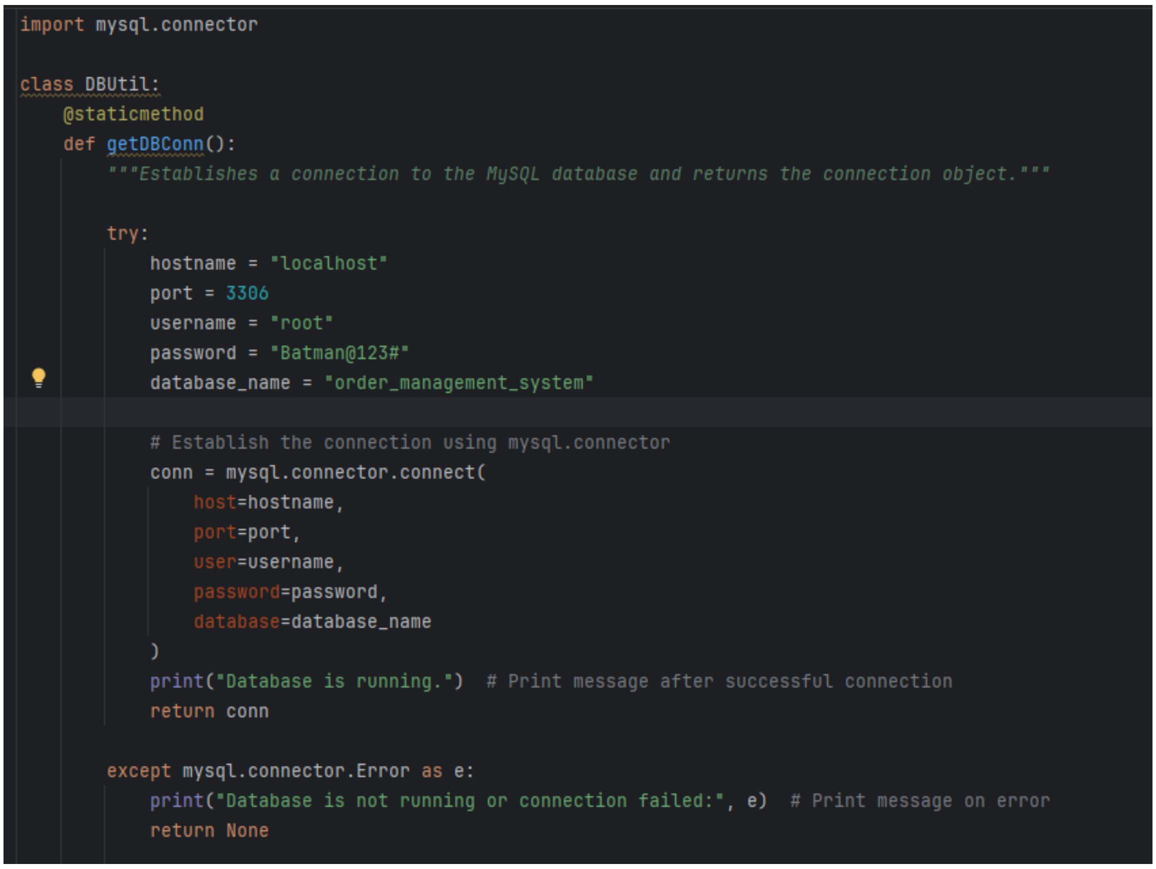
- *TASK 7 :Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.*



```
from abc import ABC, abstractmethod
from i_order_management_repository import IOrderManagementRepository
class OrderProcessor(IOrderManagementRepository):
    def __init__(self, db_conn):
        self.db_conn = db_conn

    def create_order(self, user, products):
        # Implement order creation logic using the database connection
        # Example using placeholders:
        cursor = self.db_conn.cursor()
        cursor.execute("INSERT INTO orders (user_id, products) VALUES (%s, %s)", (user.id, products))
        self.db_conn.commit()
        return True  # Replace with actual order object or ID

    1 usage (1 dynamic)
    def cancel_order(self, user_id, order_id):
        # Implement order cancellation logic
        cursor = self.db_conn.cursor()
        cursor.execute("UPDATE orders SET status = 'canceled' WHERE user_id = %s AND id = %s", (user_id, order_id))
        self.db_conn.commit()
        return True

    # Implement other methods similarly using database operations

    1 usage (1 dynamic)
    def create_product(self, user, product):
        pass

    1 usage (1 dynamic)
    def create_user(self, user):
        pass

    1 usage (1 dynamic)
    def get_all_products(self):
        pass

    1 usage (1 dynamic)
    def get_orders_by_user(self, user):
        pass
```

- *TASK 8 :Create DBUtil class and add the following method.*

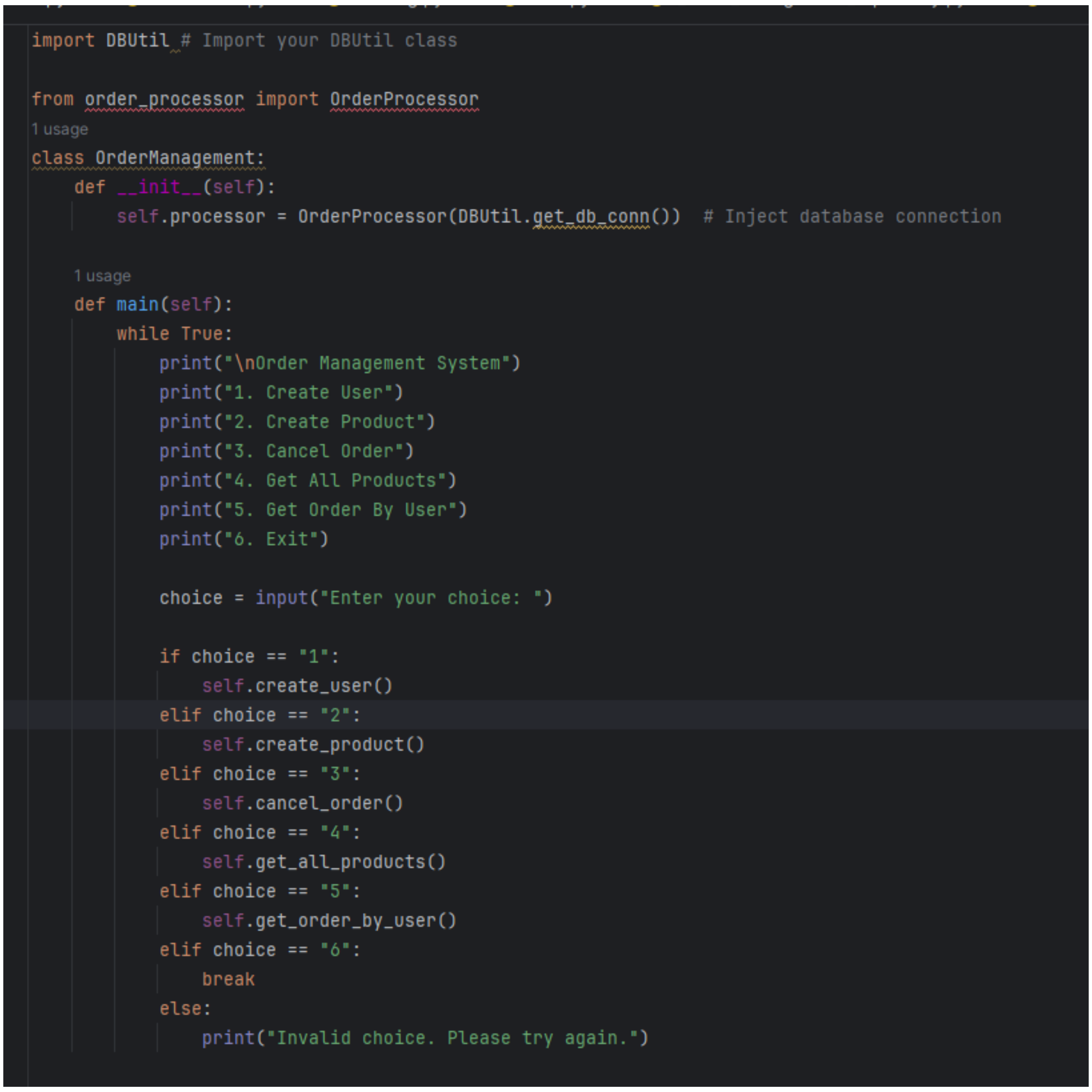
  • static getDBConn():Connection Establish a connection to the database and return database Connection



```python
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        """Establishes a connection to the MySQL database and returns the connection object."""

        try:
            hostname = "localhost"
            port = 3306
            username = "root"
            password = "Batman@123#"
            database_name = "order_management_system"

            # Establish the connection using mysql.connector
            conn = mysql.connector.connect(
                host=hostname,
                port=port,
                user=username,
                password=password,
                database=database_name
            )
            print("Database is running.")  # Print message after successful connection
            return conn

        except mysql.connector.Error as e:
            print("Database is not running or connection failed:", e)  # Print message on error
            return None
```

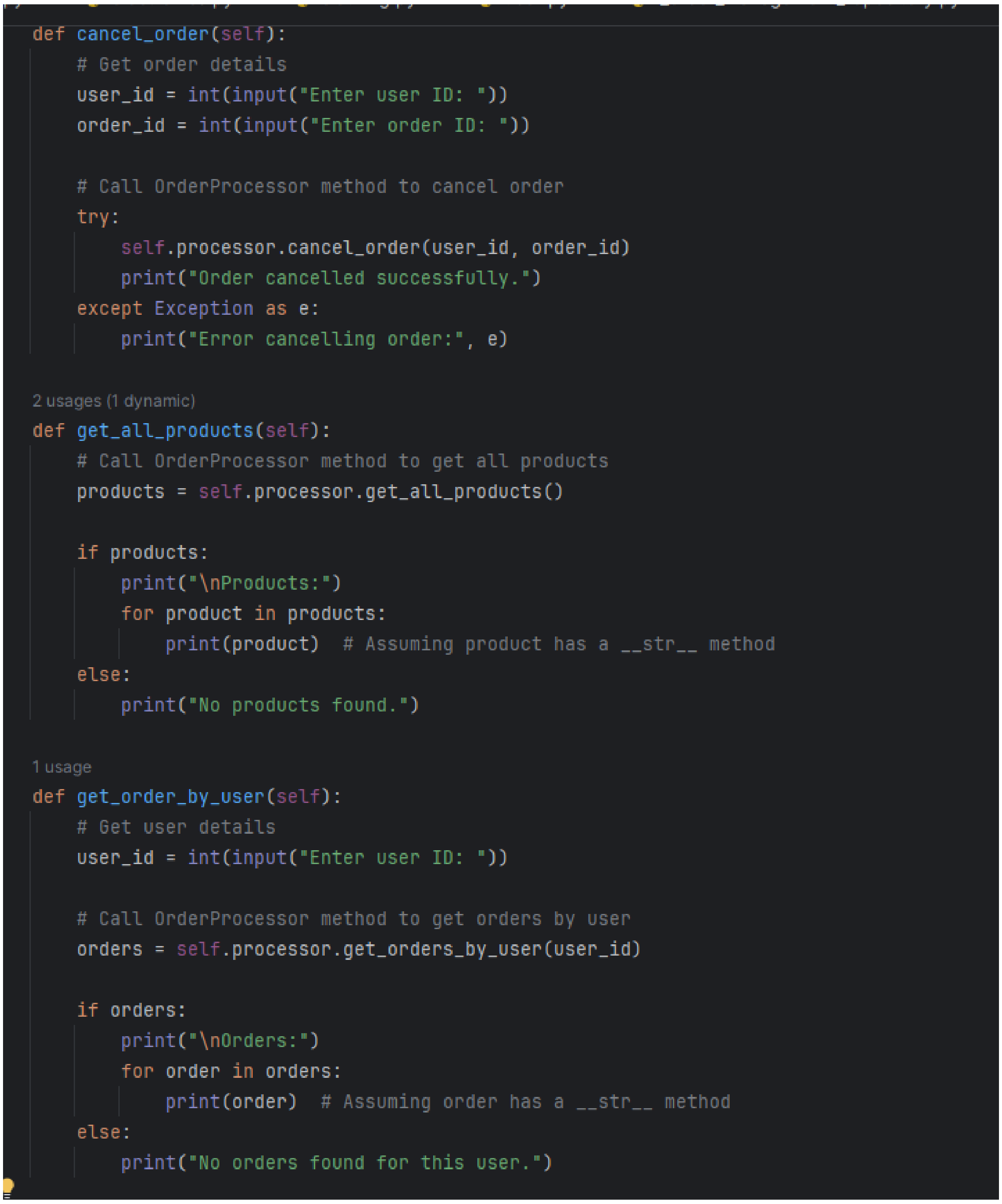
- ***<u>TASK 9 :Create OrderManagement main class and perform following operation:</u>***

  • main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct","cancelOrder", "getAllProducts", "getOrderbyUser", "exit".

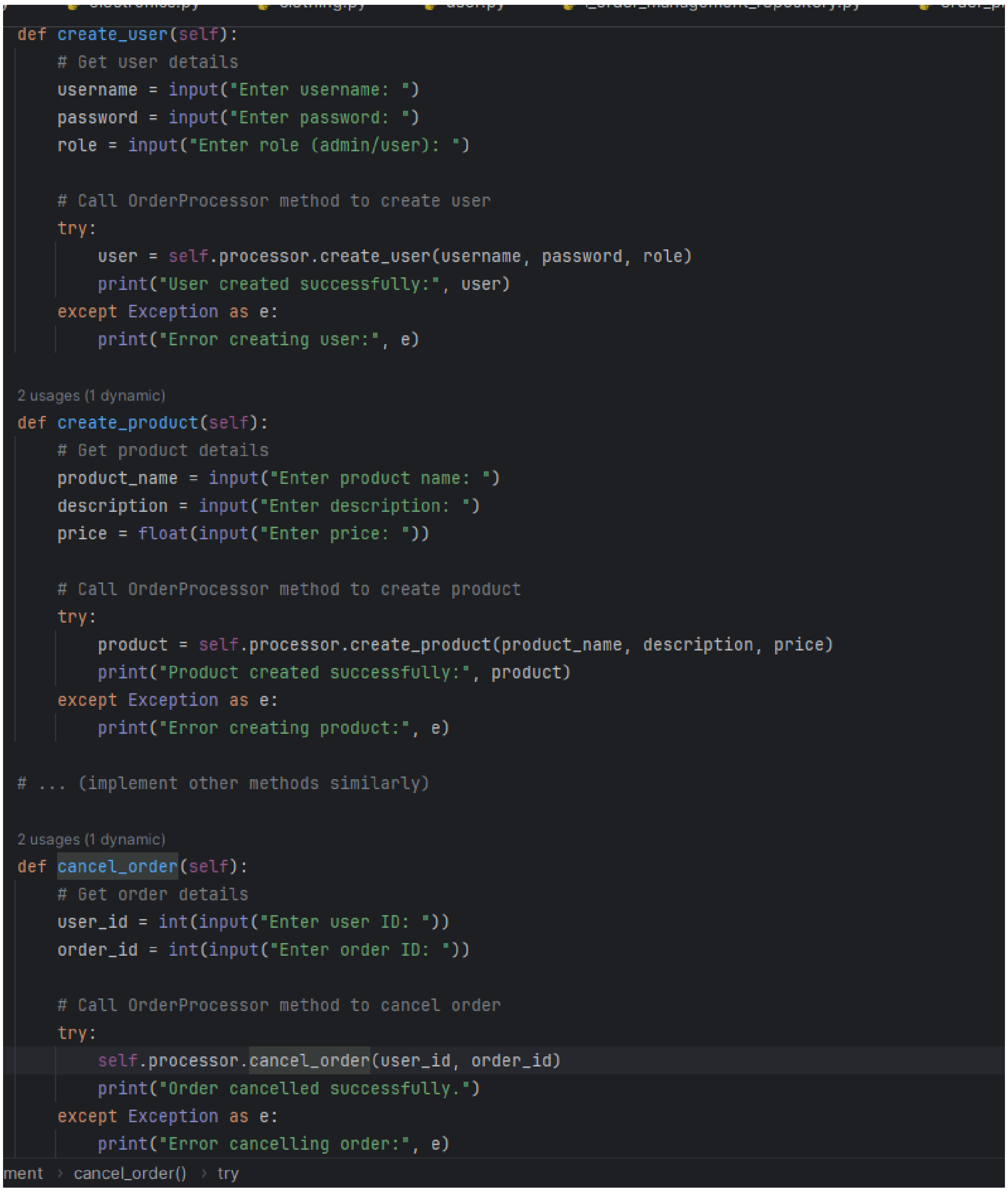


```
import DBUtil # Import your DBUtil class

from order_processor import OrderProcessor
1 usage
class OrderManagement:
    def __init__(self):
        self.processor = OrderProcessor(DBUtil.get_db_conn())  # Inject database connection

    1 usage
    def main(self):
        while True:
            print("\nOrder Management System")
            print("1. Create User")
            print("2. Create Product")
            print("3. Cancel Order")
            print("4. Get All Products")
            print("5. Get Order By User")
            print("6. Exit")

            choice = input("Enter your choice: ")

            if choice == "1":
                self.create_user()
            elif choice == "2":
                self.create_product()
            elif choice == "3":
                self.cancel_order()
            elif choice == "4":
                self.get_all_products()
            elif choice == "5":
                self.get_order_by_user()
            elif choice == "6":
                break
            else:
                print("Invalid choice. Please try again.")
```

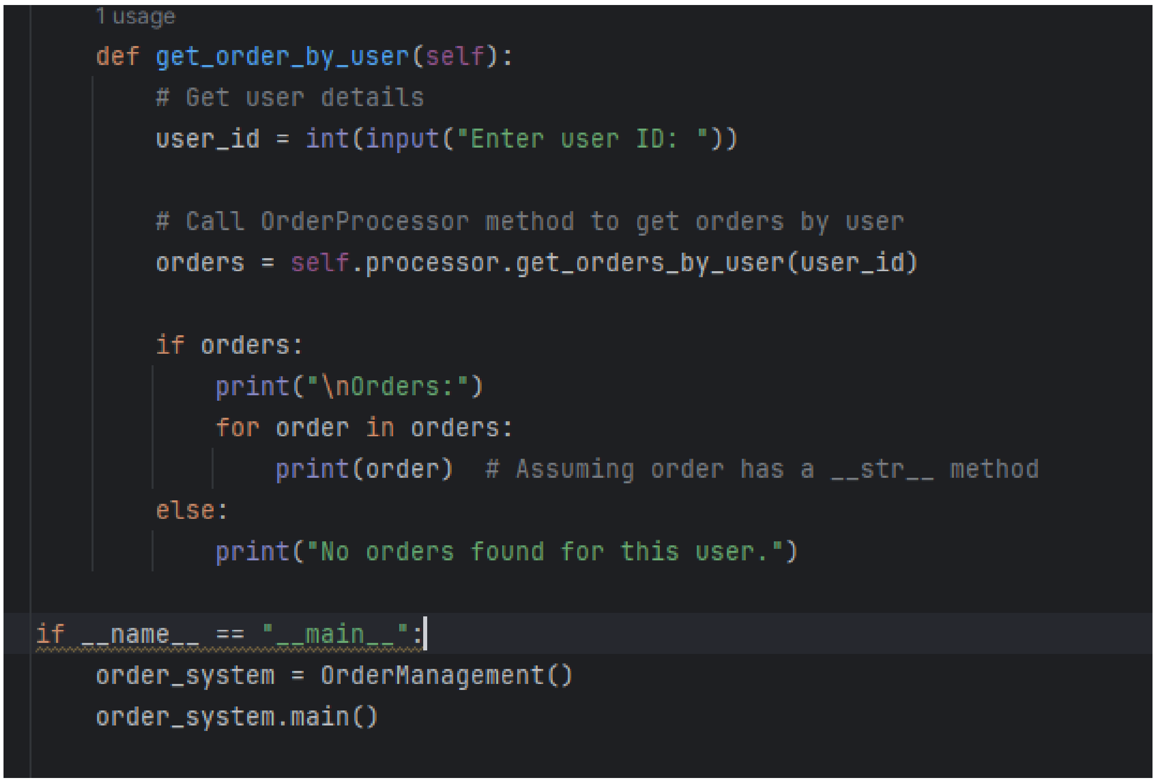
- ++



```
def cancel_order(self):
    # Get order details
    user_id = int(input("Enter user ID: "))
    order_id = int(input("Enter order ID: "))

    # Call OrderProcessor method to cancel order
    try:
        self.processor.cancel_order(user_id, order_id)
        print("Order cancelled successfully.")
    except Exception as e:
        print("Error cancelling order:", e)

2 usages (1 dynamic)
def get_all_products(self):
    # Call OrderProcessor method to get all products
    products = self.processor.get_all_products()

    if products:
        print("\nProducts:")
        for product in products:
            print(product)  # Assuming product has a __str__ method
    else:
        print("No products found.")

1 usage
def get_order_by_user(self):
    # Get user details
    user_id = int(input("Enter user ID: "))

    # Call OrderProcessor method to get orders by user
    orders = self.processor.get_orders_by_user(user_id)

    if orders:
        print("\nOrders:")
        for order in orders:
            print(order)  # Assuming order has a __str__ method
    else:
        print("No orders found for this user.")
```

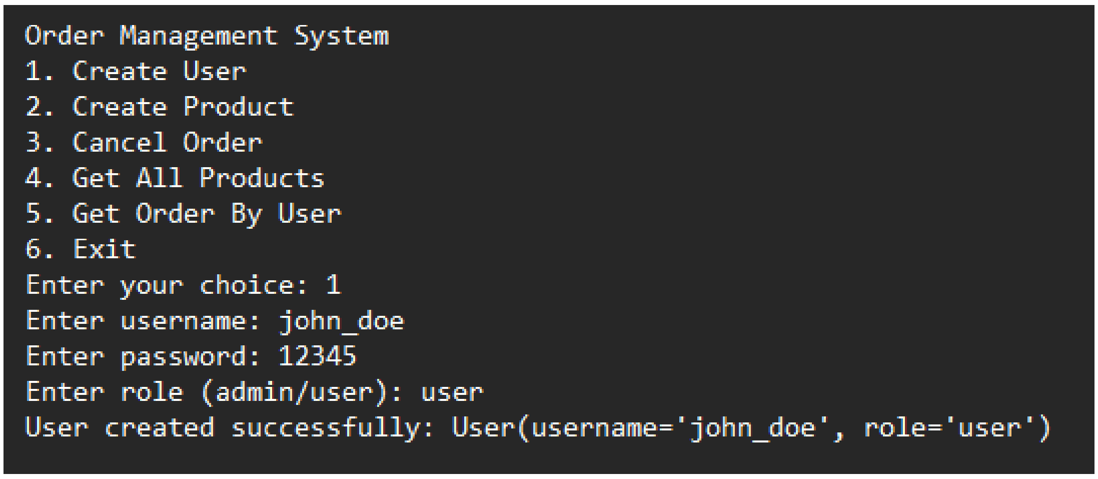


```
def create_user(self):
    # Get user details
    username = input("Enter username: ")
    password = input("Enter password: ")
    role = input("Enter role (admin/user): ")

    # Call OrderProcessor method to create user
    try:
        user = self.processor.create_user(username, password, role)
        print("User created successfully:", user)
    except Exception as e:
        print("Error creating user:", e)

2 usages (1 dynamic)
def create_product(self):
    # Get product details
    product_name = input("Enter product name: ")
    description = input("Enter description: ")
    price = float(input("Enter price: "))

    # Call OrderProcessor method to create product
    try:
        product = self.processor.create_product(product_name, description, price)
        print("Product created successfully:", product)
    except Exception as e:
        print("Error creating product:", e)

# ... (implement other methods similarly)

2 usages (1 dynamic)
def cancel_order(self):
    # Get order details
    user_id = int(input("Enter user ID: "))
    order_id = int(input("Enter order ID: "))

    # Call OrderProcessor method to cancel order
    try:
        self.processor.cancel_order(user_id, order_id)
        print("Order cancelled successfully.")
    except Exception as e:
        print("Error cancelling order:", e)
```



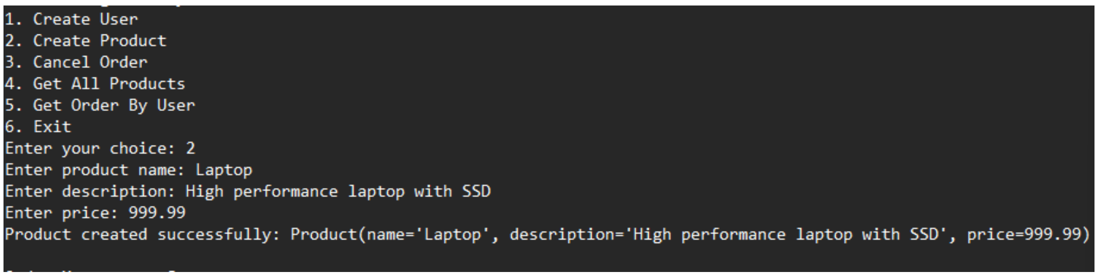
```
    1 usage
    def get_order_by_user(self):
        # Get user details
        user_id = int(input("Enter user ID: "))

        # Call OrderProcessor method to get orders by user
        orders = self.processor.get_orders_by_user(user_id)

        if orders:
            print("\nOrders:")
            for order in orders:
                print(order)  # Assuming order has a __str__ method
        else:
            print("No orders found for this user.")

if __name__ == "__main__":
    order_system = OrderManagement()
    order_system.main()
```

- ***Conclusion :***
    - *This project implemented an Order Management System using object-oriented principles, SQL, and database interaction. It features user-defined exceptions, encapsulation, and a menu-driven interface, showcasing essential functionalities like user and product management, order creation, and cancellation.*
- Output Screen after running the main module
    - 1.CREATE USER



```
Order Management System
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Order By User
6. Exit
Enter your choice: 1
Enter username: john_doe
Enter password: 12345
Enter role (admin/user): user
User created successfully: User(username='john_doe', role='user')
```

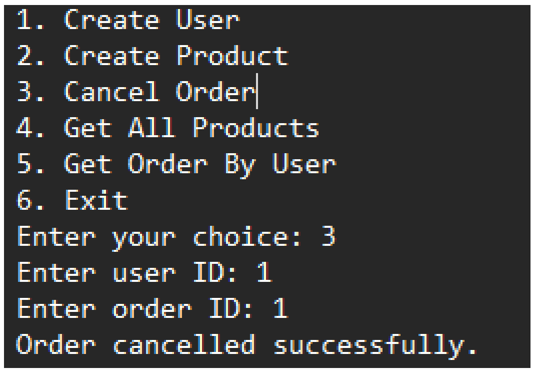
    - 2.



```
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Order By User
6. Exit
Enter your choice: 2
Enter product name: Laptop
Enter description: High performance laptop with SSD
Enter price: 999.99
Product created successfully: Product(name='Laptop', description='High performance laptop with SSD', price=999.99)
```

  - CANCEL ORDER

- 3.get all products



```
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Order By User
6. Exit
Enter your choice: 3
Enter user ID: 1
Enter order ID: 1
Order cancelled successfully.
```