

Coding Challenge:-6

By SHREYASI REJA

Coding Challenge - Order Management System

Problem Statement:

Create SQL Schema from the product and user class, use the class attributes for table column names.

1. Create a base class called Product with the following attributes:

productId (int)

productName (String)

description (String)

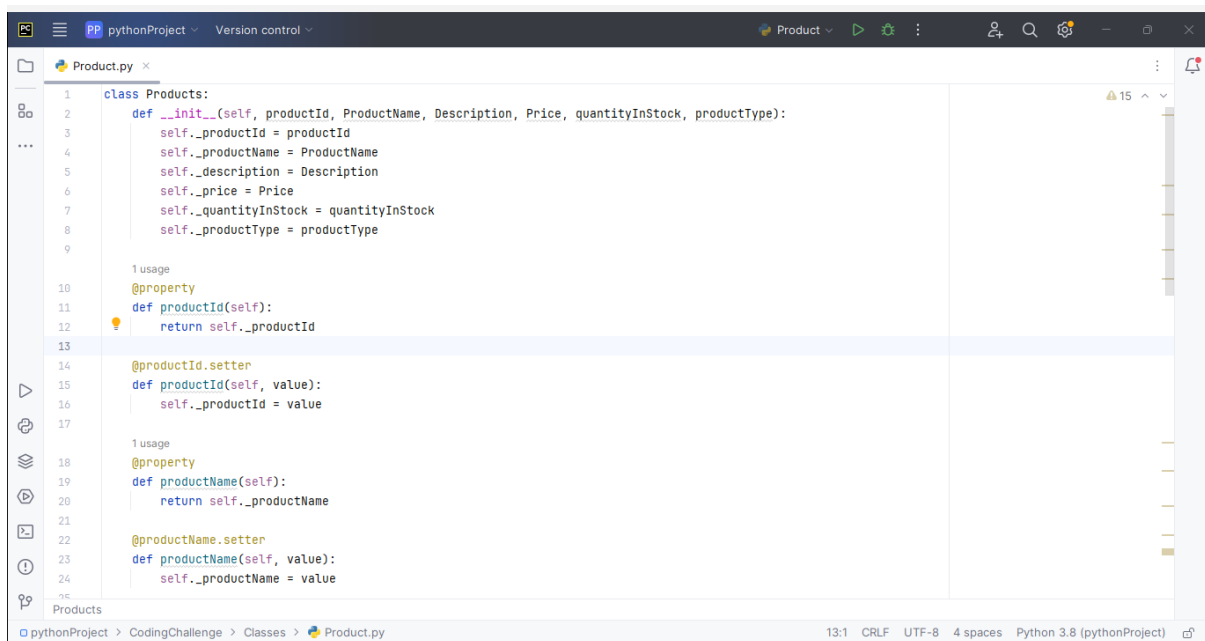
price (double)

quantityInStock (int)

type (String) [Electronics/Clothing]

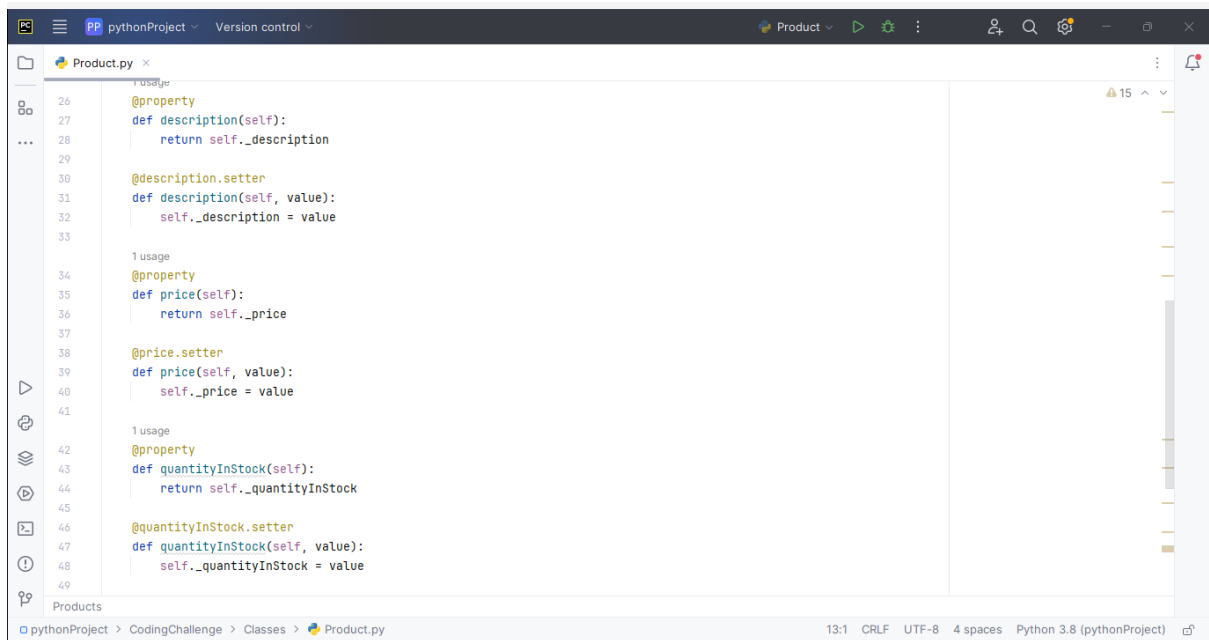
2. Implement constructors, getters, and setters for the Product class.

CODE:-

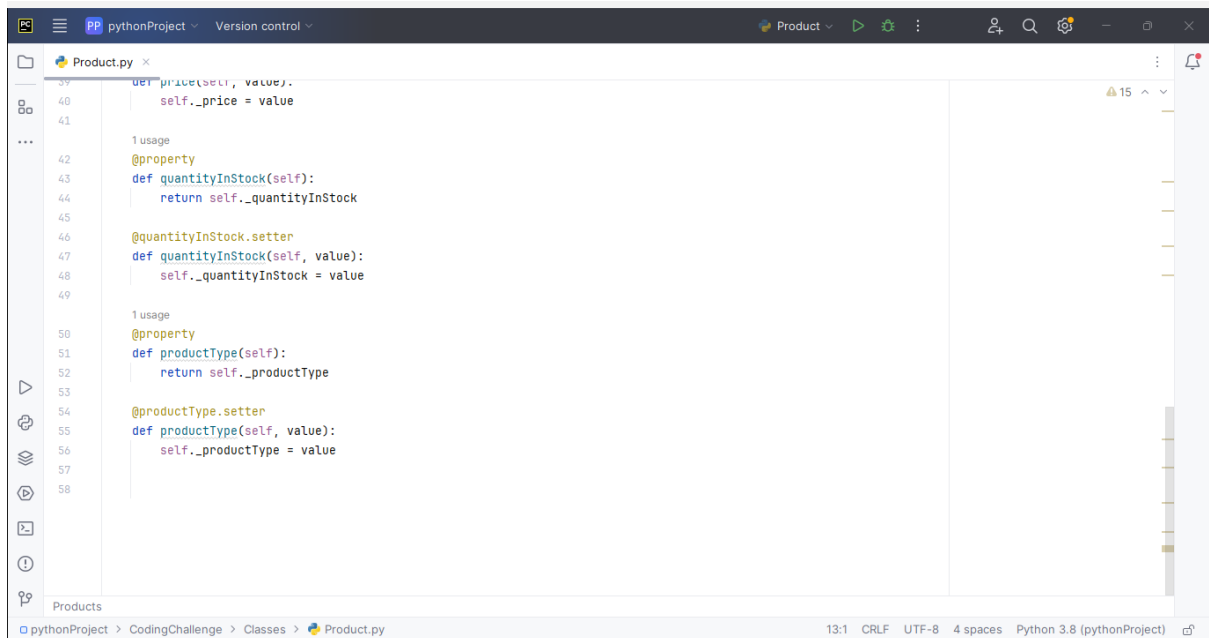
A screenshot of a code editor window titled 'pythonProject' showing the implementation of a 'Product' class in Python. The code is as follows:

```
1 class Product:
2     def __init__(self, productId, ProductName, Description, Price, quantityInStock, productType):
3         self.__productId = productId
4         self.__productName = ProductName
5         self.__description = Description
6         self.__price = Price
7         self.__quantityInStock = quantityInStock
8         self.__productType = productType
9
10    1 usage
11    @property
12    def productId(self):
13        return self.__productId
14
15    @productId.setter
16    def productId(self, value):
17        self.__productId = value
18
19    1 usage
20    @property
21    def productName(self):
22        return self.__productName
23
24    @productName.setter
25    def productName(self, value):
26        self.__productName = value
```

The editor interface includes a sidebar on the left with icons for file explorer, search, and other IDE features. The bottom status bar shows '13:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)'.



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



```
39 def price(self, value):
40     self._price = value
41
42 1 usage
43 @property
44 def quantityInStock(self):
45     return self._quantityInStock
46
47 @quantityInStock.setter
48 def quantityInStock(self, value):
49     self._quantityInStock = value
50
51 1 usage
52 @property
53 def productType(self):
54     return self._productType
55
56 @productType.setter
57 def productType(self, value):
58     self._productType = value
```

3. Create a subclass Electronics that inherits from Product. Add attributes specific to electronics products, such as:

brand (String)

warrantyPeriod (int)

code:-

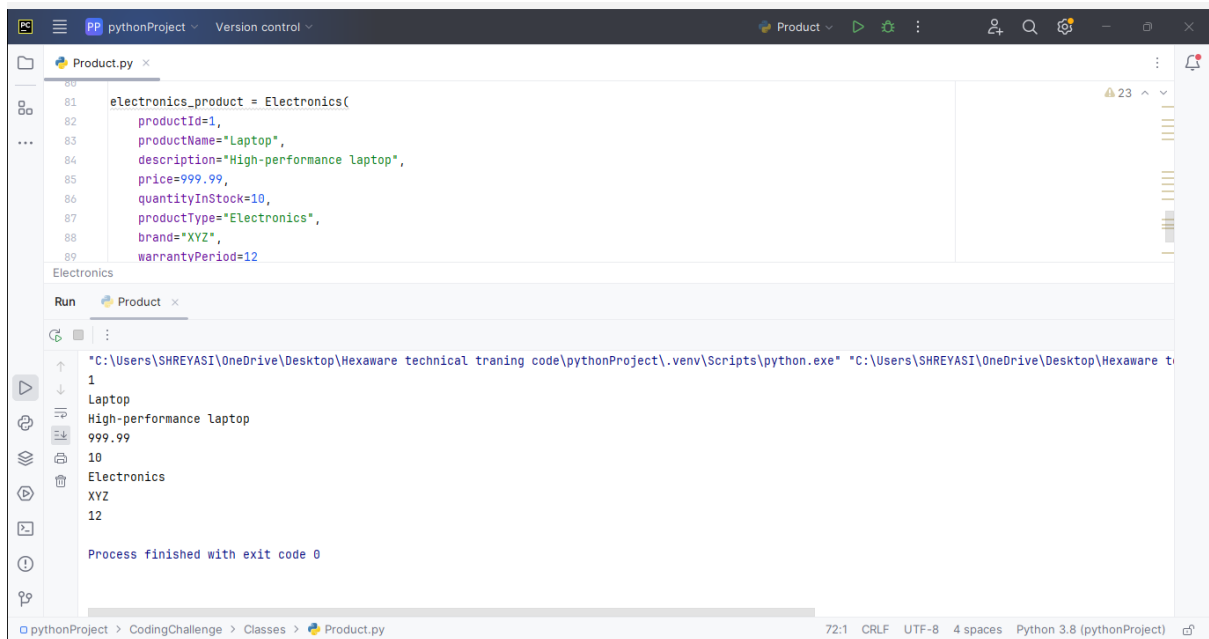
The screenshot shows a code editor window for a file named `Product.py`. The code defines a class `Electronics` with an `__init__` method and two properties: `brand` and `warrantyPeriod`. The `__init__` method takes parameters `productId`, `productName`, `description`, `price`, `quantityInStock`, `productType`, `brand`, and `warrantyPeriod`. The `brand` property has a getter and a setter. The `warrantyPeriod` property also has a getter and a setter. At the bottom, an instance `electronics_product` is created with `productId=1`.

```
59 class Electronics(Products):
60     def __init__(self, productId, productName, description, price, quantityInStock, productType, brand, warrantyPeriod):
61         super().__init__(productId, productName, description, price, quantityInStock, productType)
62         self._brand = brand
63         self._warrantyPeriod = warrantyPeriod
64
65     2 usages
66     @property
67     def brand(self):
68         return self._brand
69
70     @brand.setter
71     def brand(self, value):
72         self._brand = value
73
74     2 usages
75     @property
76     def warrantyPeriod(self):
77         return self._warrantyPeriod
78
79     @warrantyPeriod.setter
80     def warrantyPeriod(self, value):
81         self._warrantyPeriod = value
82
83 electronics_product = Electronics(
84     productId=1,
```

The screenshot shows the continuation of the `Product.py` file. The `electronics_product` instance is fully instantiated with specific values for all attributes. Below the instantiation, there are ten `print` statements that output the values of each attribute: `productId`, `productName`, `description`, `price`, `quantityInStock`, `productType`, `brand`, and `warrantyPeriod`.

```
81 electronics_product = Electronics(
82     productId=1,
83     productName="Laptop",
84     description="High-performance laptop",
85     price=999.99,
86     quantityInStock=10,
87     productType="Electronics",
88     brand="XYZ",
89     warrantyPeriod=12
90 )
91
92 print(electronics_product.productId)
93 print(electronics_product.productName)
94 print(electronics_product.description)
95 print(electronics_product.price)
96 print(electronics_product.quantityInStock)
97 print(electronics_product.productType)
98 print(electronics_product.brand)
99 print(electronics_product.warrantyPeriod)
100
101
```

Output:-



```
80
81 electronics_product = Electronics(
82     productId=1,
83     productName="Laptop",
84     description="High-performance laptop",
85     price=999.99,
86     quantityInStock=10,
87     productType="Electronics",
88     brand="XYZ",
89     warrantyPeriod=12
90 )
91
92 # Print product details
93 print(electronics_product.productId)
94 print(electronics_product.productName)
95 print(electronics_product.description)
96 print(electronics_product.price)
97 print(electronics_product.quantityInStock)
98 print(electronics_product.productType)
99 print(electronics_product.brand)
100 print(electronics_product.warrantyPeriod)
```

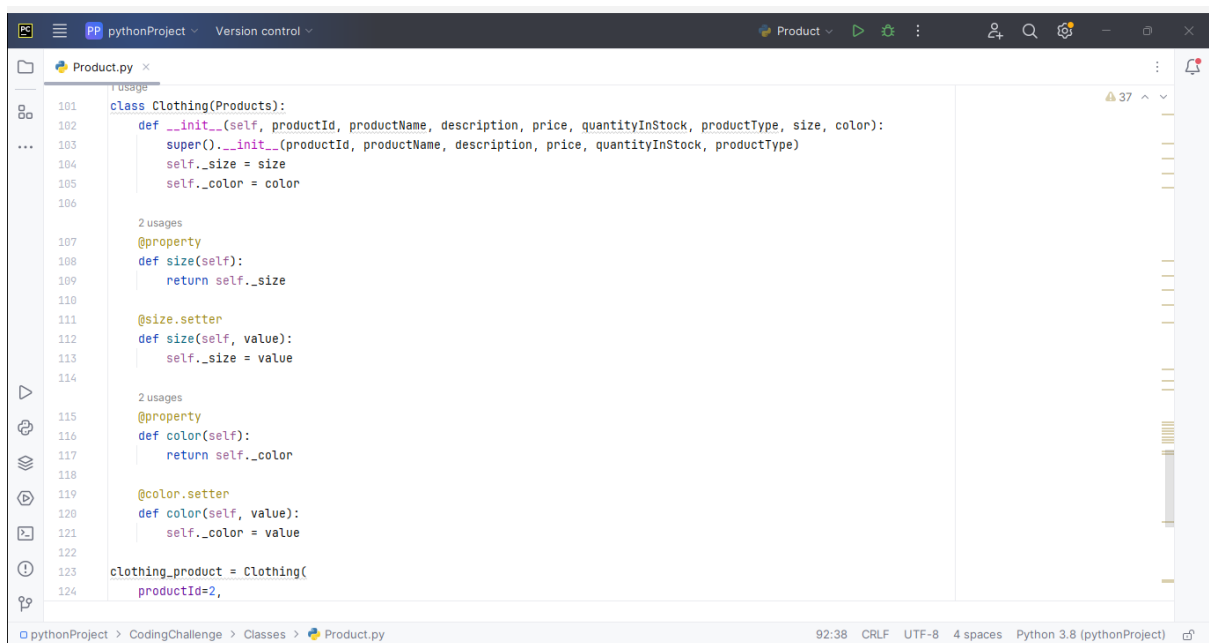
Run Product

```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical traning code\pythonProject\.venv\Scripts\python.exe *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
1
Laptop
High-performance laptop
999.99
10
Electronics
XYZ
12
Process finished with exit code 0
```

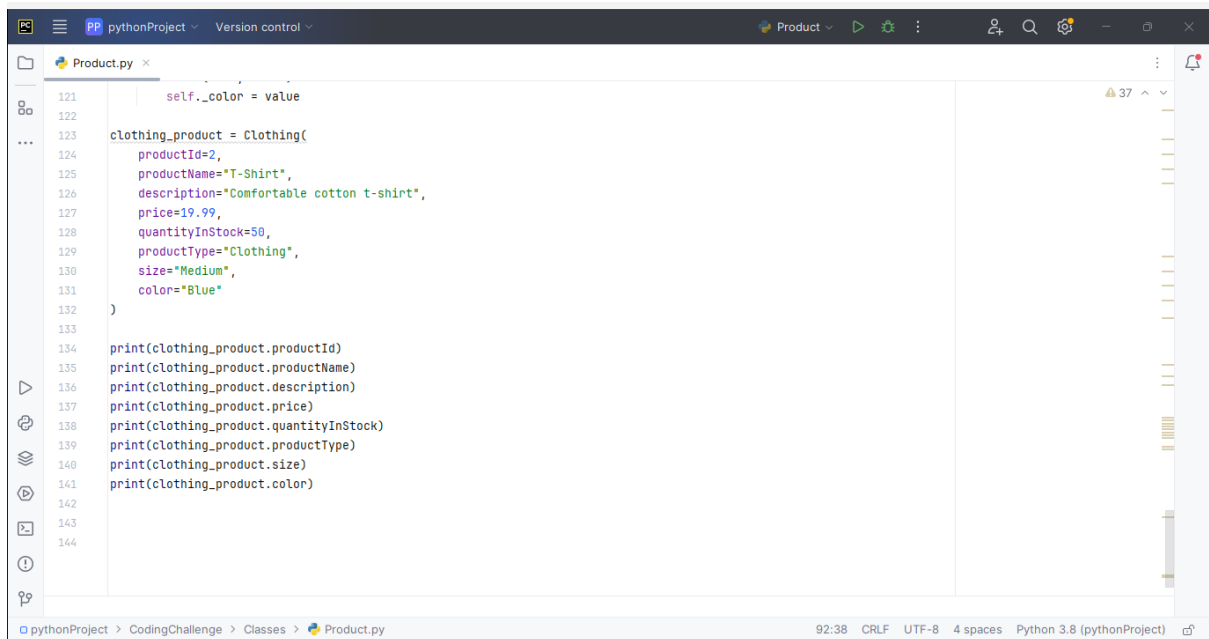
4. Create a subclass Clothing that also inherits from Product. Add attributes specific to clothing products, such as:

size (String)

color (String)

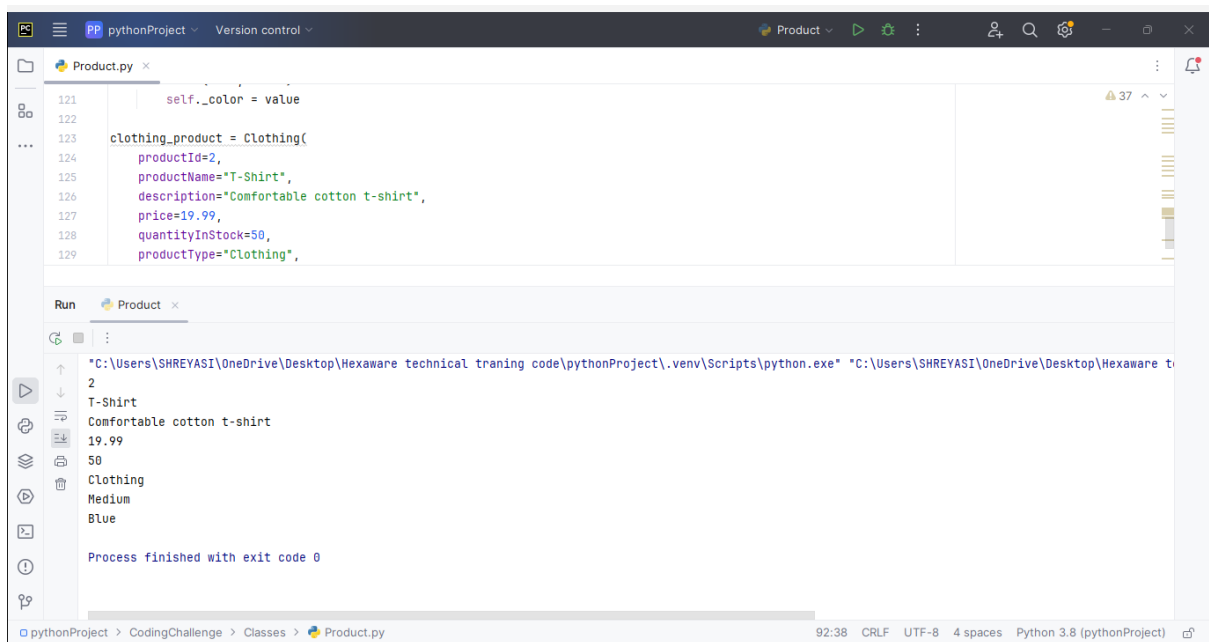


```
101 class Clothing(Product):
102     def __init__(self, productId, productName, description, price, quantityInStock, productType, size, color):
103         super().__init__(productId, productName, description, price, quantityInStock, productType)
104         self._size = size
105         self._color = color
106
107     2 usages
108     @property
109     def size(self):
110         return self._size
111
112     @size.setter
113     def size(self, value):
114         self._size = value
115
116     2 usages
117     @property
118     def color(self):
119         return self._color
120
121     @color.setter
122     def color(self, value):
123         self._color = value
124
125 clothing_product = Clothing(
126     productId=2,
```



```
121         self._color = value
122
123     clothing_product = Clothing(
124         productId=2,
125         productName="T-Shirt",
126         description="Comfortable cotton t-shirt",
127         price=19.99,
128         quantityInStock=50,
129         productType="Clothing",
130         size="Medium",
131         color="Blue"
132     )
133
134     print(clothing_product.productId)
135     print(clothing_product.productName)
136     print(clothing_product.description)
137     print(clothing_product.price)
138     print(clothing_product.quantityInStock)
139     print(clothing_product.productType)
140     print(clothing_product.size)
141     print(clothing_product.color)
142
143
144
```

Output:-



```
*C:\Users\SHREYASI\OneDrive\Desktop\Hexaware technical tranning code\pythonProject\.venv\Scripts\python.exe* *C:\Users\SHREYASI\OneDrive\Desktop\Hexaware t
2
T-Shirt
Comfortable cotton t-shirt
19.99
50
Clothing
Medium
Blue

Process finished with exit code 0
```

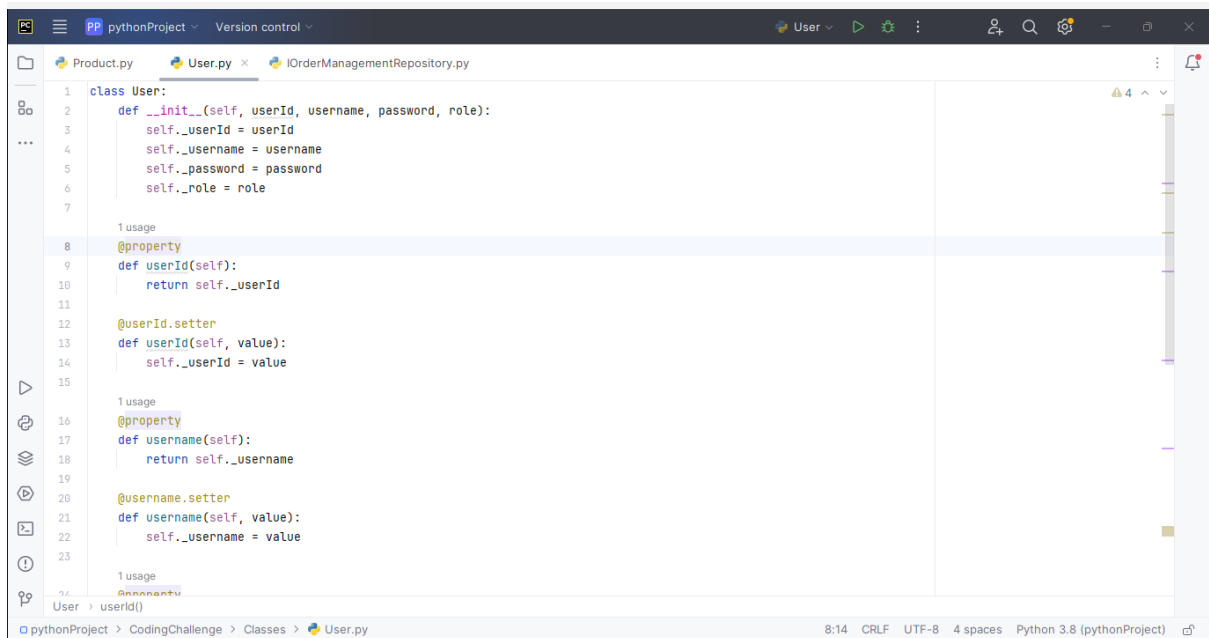
5. Create a User class with attributes:

userId (int)

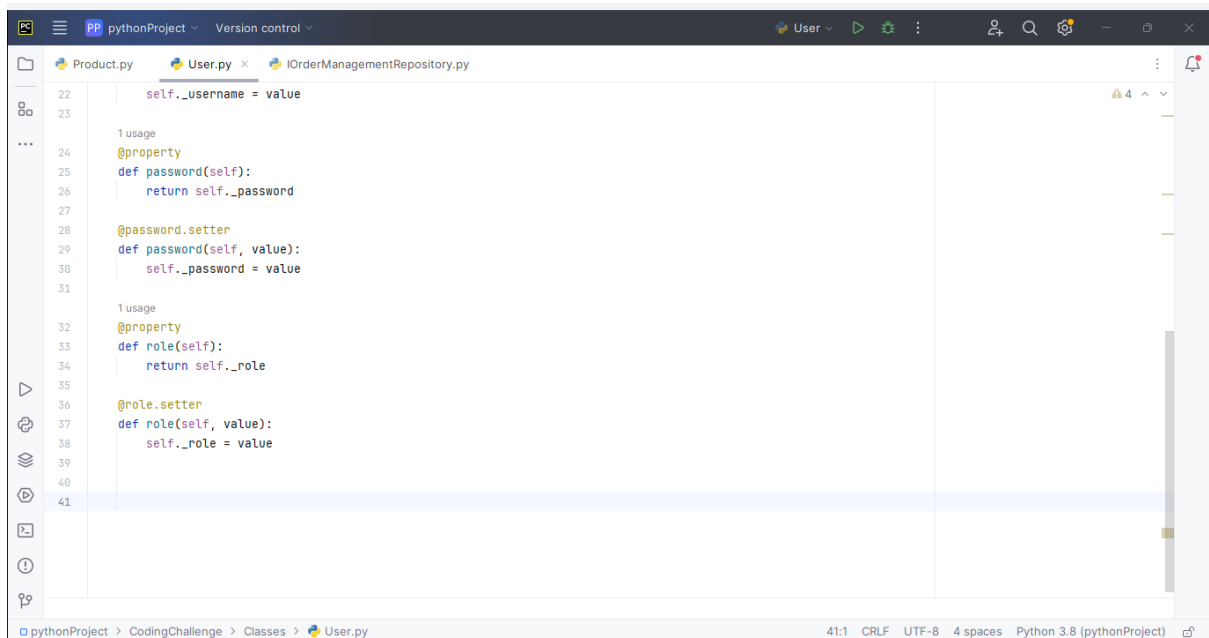
username (String)

password (String)

role (String) // "Admin" or "User"



```
1 class User:
2     def __init__(self, userId, username, password, role):
3         self._userId = userId
4         self._username = username
5         self._password = password
6         self._role = role
7
8     @property
9     def userId(self):
10         return self._userId
11
12     @userId.setter
13     def userId(self, value):
14         self._userId = value
15
16     @property
17     def username(self):
18         return self._username
19
20     @username.setter
21     def username(self, value):
22         self._username = value
23
24     @property
25     def password(self):
26         return self._password
27
28     @password.setter
29     def password(self, value):
30         self._password = value
31
32     @property
33     def role(self):
34         return self._role
35
36     @role.setter
37     def role(self, value):
38         self._role = value
39
40
41
```



```
22         self._username = value
23
24     @property
25     def password(self):
26         return self._password
27
28     @password.setter
29     def password(self, value):
30         self._password = value
31
32     @property
33     def role(self):
34         return self._role
35
36     @role.setter
37     def role(self, value):
38         self._role = value
39
40
41
```

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

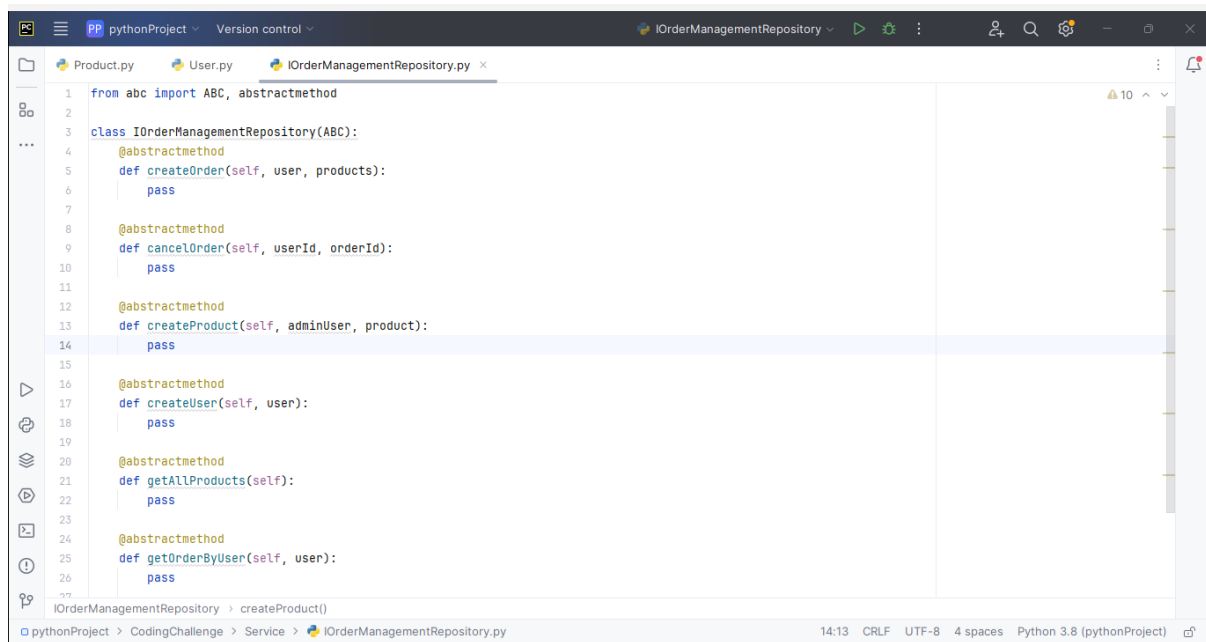
createProduct(User user, Product product): check the admin user as already present in database and create product and store in database.

createUser(User user): create user and store in database for further development.

getAllProducts(): return all product list from the database.

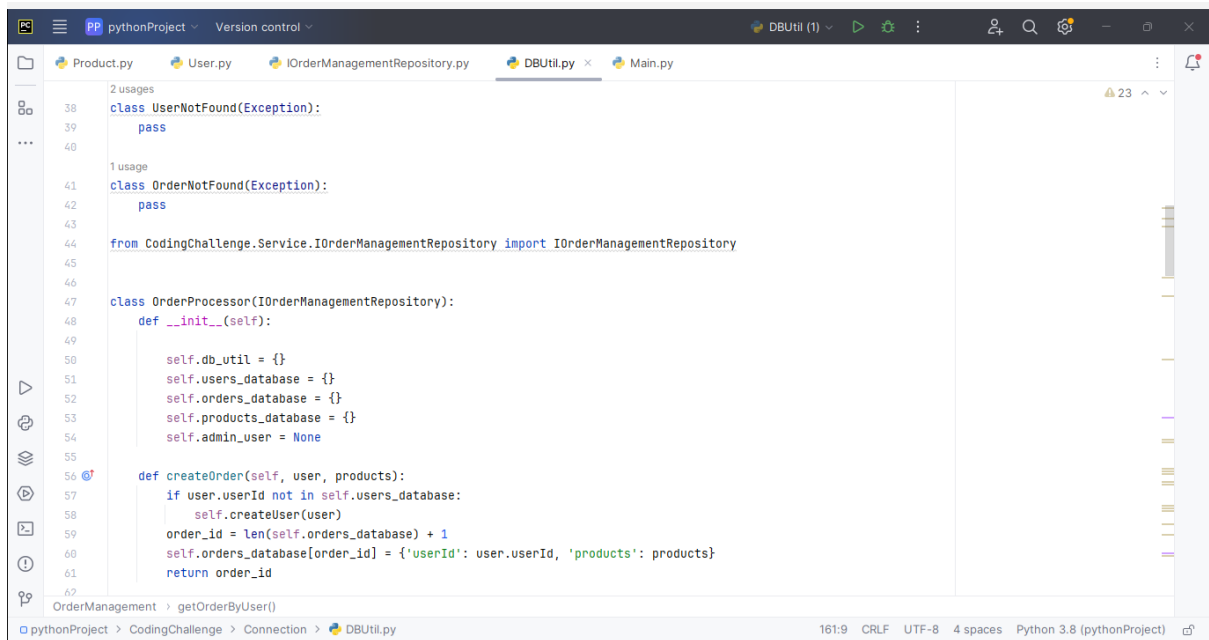
getOrderByUser(User user): return all product ordered by specific user from database.

Code:-

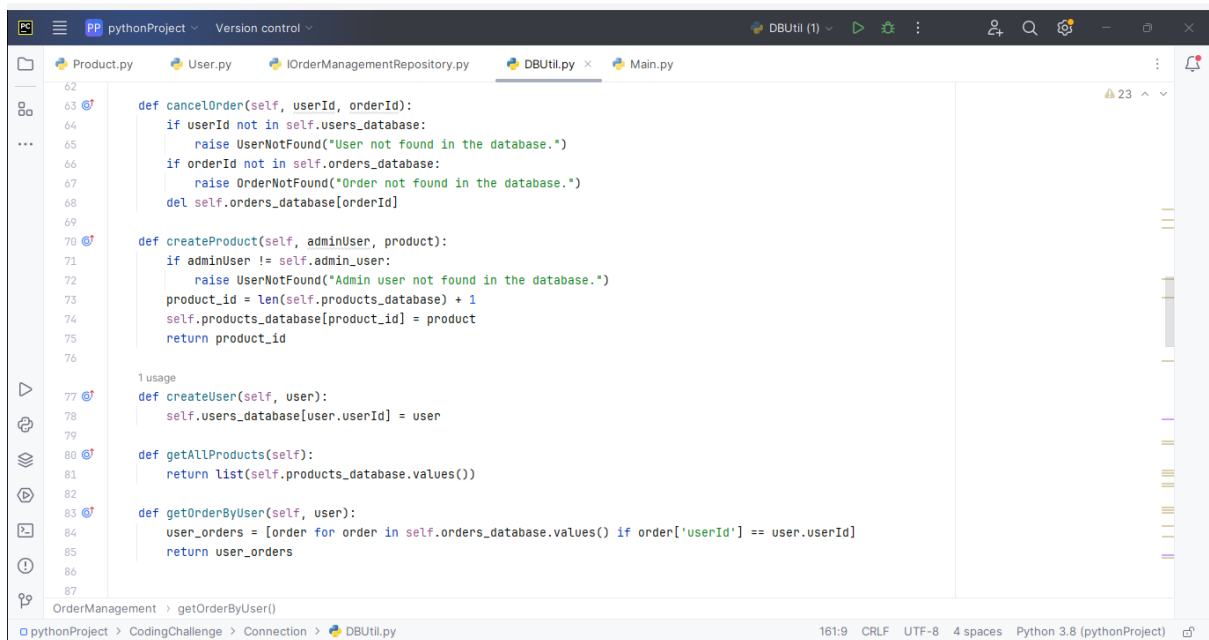
A screenshot of a code editor window showing a Python file named 'IOrderManagementRepository.py'. The code defines an abstract class 'IOrderManagementRepository' that inherits from 'ABC' and implements several abstract methods. The methods are: 'createOrder', 'cancelOrder', 'createProduct', 'createUser', 'getAllProducts', and 'getOrderByUser'. Each method is decorated with '@abstractmethod' and contains a 'pass' statement. The editor interface includes a sidebar with file explorer and search icons, a top bar with project and file names, and a bottom status bar showing file encoding and Python version.

```
1 from abc import ABC, abstractmethod
2
3 class IOrderManagementRepository(ABC):
4     @abstractmethod
5     def createOrder(self, user, products):
6         pass
7
8     @abstractmethod
9     def cancelOrder(self, userId, orderId):
10        pass
11
12    @abstractmethod
13    def createProduct(self, adminUser, product):
14        pass
15
16    @abstractmethod
17    def createUser(self, user):
18        pass
19
20    @abstractmethod
21    def getAllProducts(self):
22        pass
23
24    @abstractmethod
25    def getOrderByUser(self, user):
26        pass
```

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



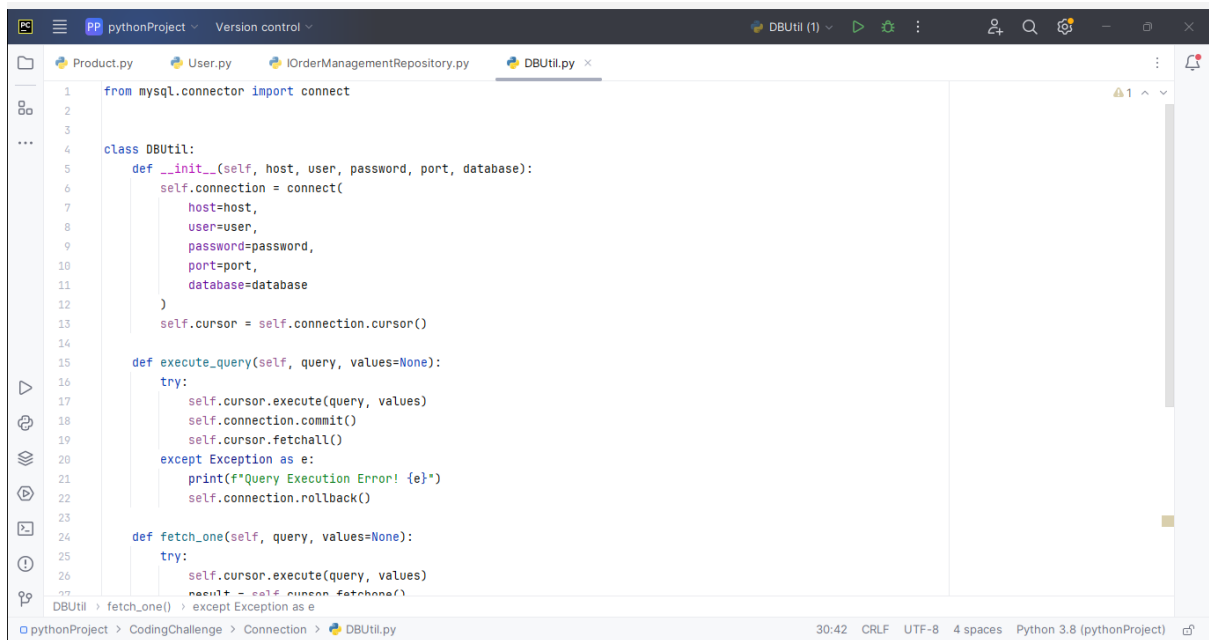
```
38 class UserNotFoundException:
39     pass
40
41 1 usage
42 class OrderNotFoundException:
43     pass
44
45 from CodingChallenge.Service.IOrderManagementRepository import IOrderManagementRepository
46
47 class OrderProcessor(IOrderManagementRepository):
48     def __init__(self):
49
50         self.db_util = {}
51         self.users_database = {}
52         self.orders_database = {}
53         self.products_database = {}
54         self.admin_user = None
55
56     def createOrder(self, user, products):
57         if user.userId not in self.users_database:
58             self.createUser(user)
59         order_id = len(self.orders_database) + 1
60         self.orders_database[order_id] = {'userId': user.userId, 'products': products}
61         return order_id
62
63 OrderManagement > getOrderByUser()
```



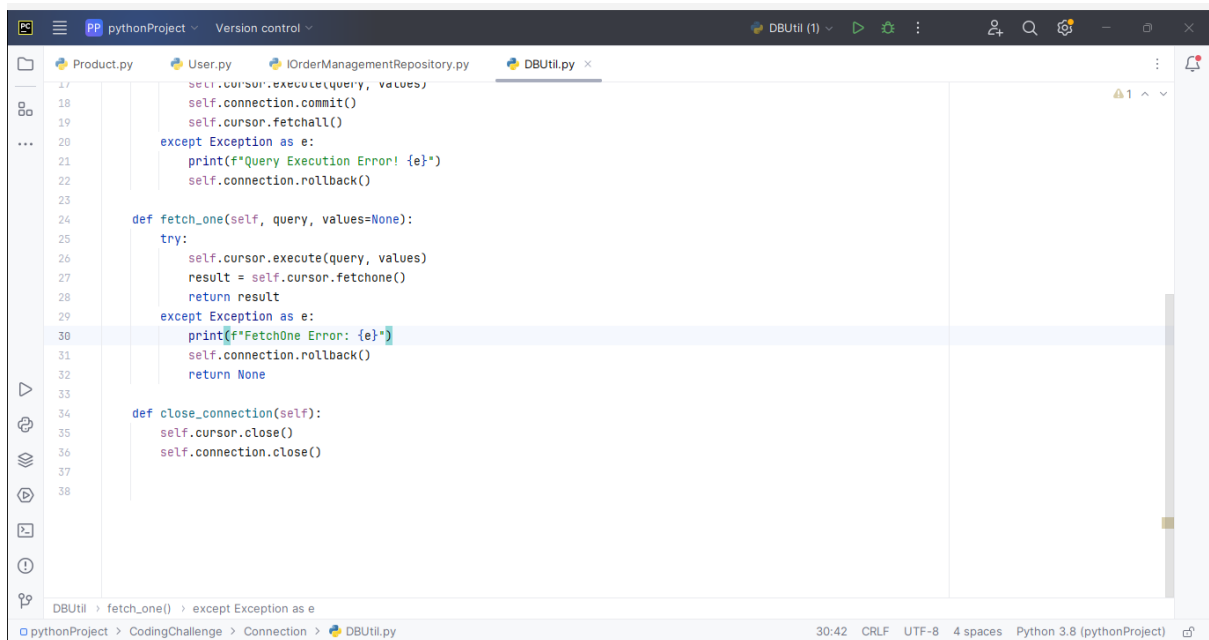
```
62
63     def cancelOrder(self, userId, orderId):
64         if userId not in self.users_database:
65             raise UserNotFoundException("User not found in the database.")
66         if orderId not in self.orders_database:
67             raise OrderNotFoundException("Order not found in the database.")
68         del self.orders_database[orderId]
69
70     def createProduct(self, adminUser, product):
71         if adminUser != self.admin_user:
72             raise UserNotFoundException("Admin user not found in the database.")
73         product_id = len(self.products_database) + 1
74         self.products_database[product_id] = product
75         return product_id
76
77 1 usage
78     def createUser(self, user):
79         self.users_database[user.userId] = user
80
81     def getAllProducts(self):
82         return list(self.products_database.values())
83
84     def getOrderByUser(self, user):
85         user_orders = [order for order in self.orders_database.values() if order['userId'] == user.userId]
86         return user_orders
87
88 OrderManagement > getOrderByUser()
```

8. Create DBUtil class and add the following method.

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



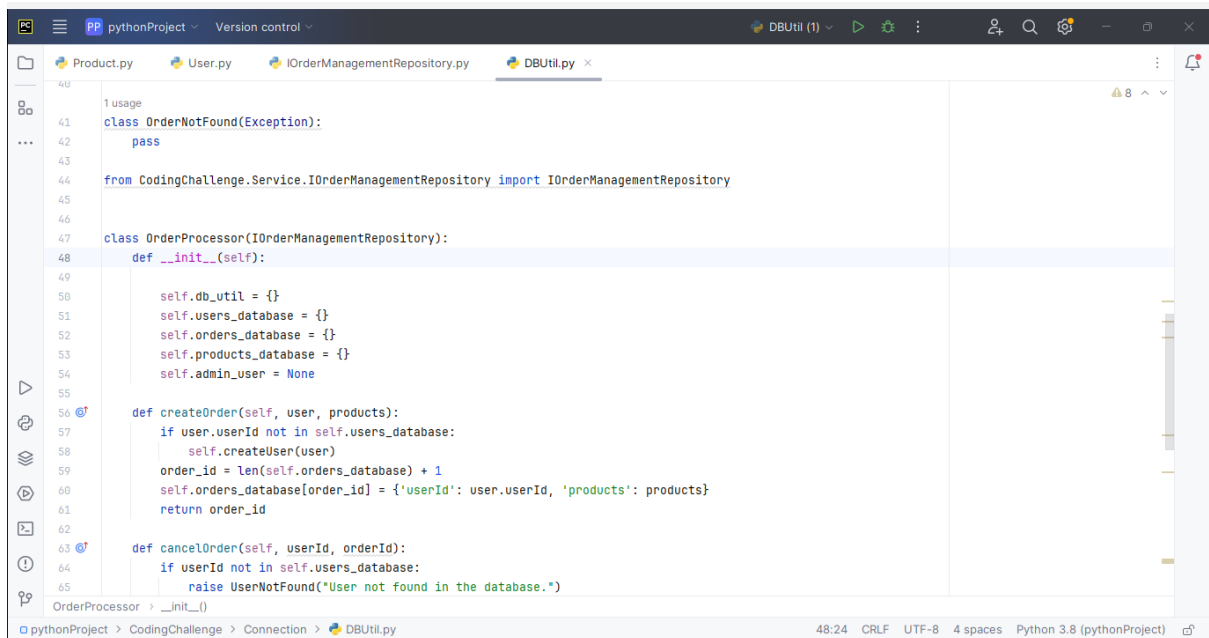
```
1 from mysql.connector import connect
2
3
4 class DBUtil:
5     def __init__(self, host, user, password, port, database):
6         self.connection = connect(
7             host=host,
8             user=user,
9             password=password,
10            port=port,
11            database=database
12        )
13        self.cursor = self.connection.cursor()
14
15    def execute_query(self, query, values=None):
16        try:
17            self.cursor.execute(query, values)
18            self.connection.commit()
19            self.cursor.fetchall()
20        except Exception as e:
21            print(f"Query Execution Error! {e}")
22            self.connection.rollback()
23
24    def fetch_one(self, query, values=None):
25        try:
26            self.cursor.execute(query, values)
27            result = self.cursor.fetchone()
28            return result
29        except Exception as e:
30            print(f"FetchOne Error: {e}")
31            self.connection.rollback()
32            return None
33
34    def close_connection(self):
35        self.cursor.close()
36        self.connection.close()
```



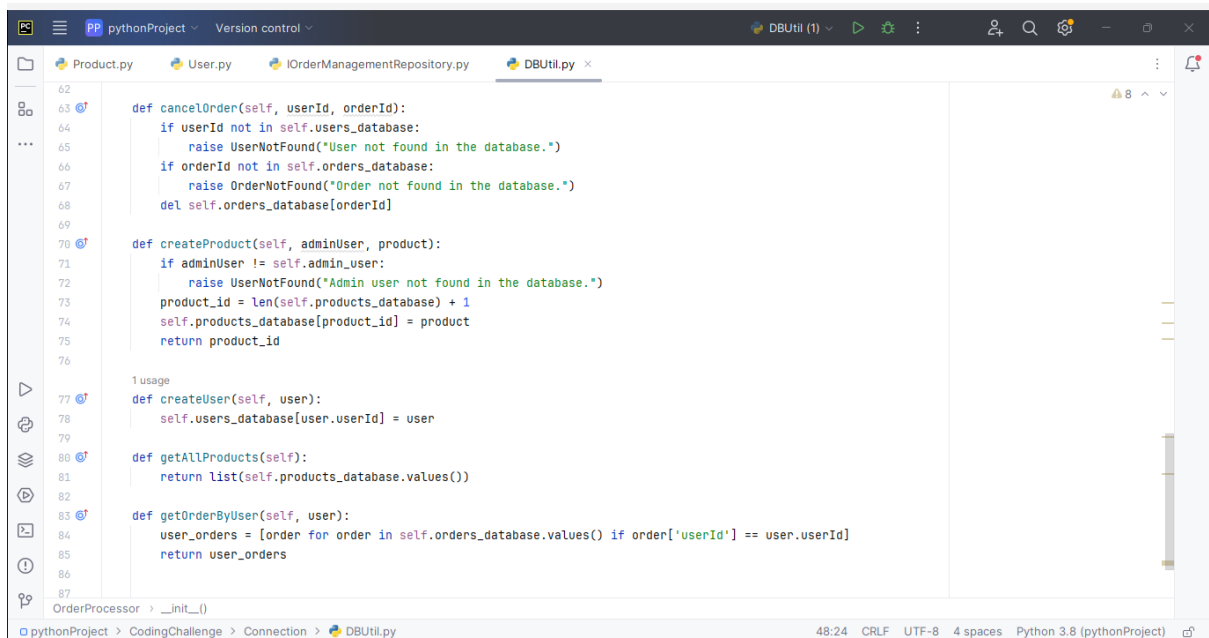
```
17 self.cursor.execute(query, values)
18 self.connection.commit()
19 self.cursor.fetchall()
20
21 except Exception as e:
22     print(f"Query Execution Error! {e}")
23     self.connection.rollback()
24
25 def fetch_one(self, query, values=None):
26     try:
27         self.cursor.execute(query, values)
28         result = self.cursor.fetchone()
29         return result
30     except Exception as e:
31         print(f"FetchOne Error: {e}")
32         self.connection.rollback()
33         return None
34
35 def close_connection(self):
36     self.cursor.close()
37     self.connection.close()
```

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

Code:-



```
40 1 usage
41 class OrderNotFound(Exception):
42     pass
43
44 from CodingChallenge.Service.IOrderManagementRepository import IOrderManagementRepository
45
46
47 class OrderProcessor(IOrderManagementRepository):
48     def __init__(self):
49
50         self.db_util = {}
51         self.users_database = {}
52         self.orders_database = {}
53         self.products_database = {}
54         self.admin_user = None
55
56     def createOrder(self, user, products):
57         if user.userId not in self.users_database:
58             self.createUser(user)
59         order_id = len(self.orders_database) + 1
60         self.orders_database[order_id] = {'userId': user.userId, 'products': products}
61         return order_id
62
63     def cancelOrder(self, userId, orderId):
64         if userId not in self.users_database:
65             raise UserNotFound("User not found in the database.")
66
67 OrderProcessor > __init__()
```



```
62
63     def cancelOrder(self, userId, orderId):
64         if userId not in self.users_database:
65             raise UserNotFound("User not found in the database.")
66         if orderId not in self.orders_database:
67             raise OrderNotFound("Order not found in the database.")
68         del self.orders_database[orderId]
69
70     def createProduct(self, adminUser, product):
71         if adminUser != self.admin_user:
72             raise UserNotFound("Admin user not found in the database.")
73         product_id = len(self.products_database) + 1
74         self.products_database[product_id] = product
75         return product_id
76
77 1 usage
78     def createUser(self, user):
79         self.users_database[user.userId] = user
80
81     def getAllProducts(self):
82         return list(self.products_database.values())
83
84     def getOrderbyUser(self, user):
85         user_orders = [order for order in self.orders_database.values() if order['userId'] == user.userId]
86         return user_orders
87
88 OrderProcessor > __init__()
```

9. Create OrderManagement main class and perform following operation:
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".

```
pythonProject Version control DBUtil (1)
Product.py User.py IOrderManagementRepository.py DBUtil.py Main.py
class OrderManagement:
    def __init__(self, order_processor):
        self.order_processor = order_processor

    def main(self):
        db_util = DBUtil(host='localhost', user='root', password='Root', port='3306', database='OrderManagementSystem')

        while True:
            print("\nMenu:")
            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.createUser()
            elif choice == '2':
                self.createProduct()
            elif choice == '3':
                self.cancelOrder()
            elif choice == '4':
                self.getAllProducts()
            elif choice == '5':
                self.getOrderByUser()
            elif choice == '6':
                break

            else:
                print("Invalid choice. Please try again.")

        1 usage
    def createUser(self):
        userId = int(input("Enter userId: "))
        username = input("Enter username: ")
        password = input("Enter password: ")
        role = input("Enter role (Admin/User): ")

        user = {'userId': userId, 'username': username, 'password': password, 'role': role}
        self.order_processor.create_user(user)
        print("User created successfully.")

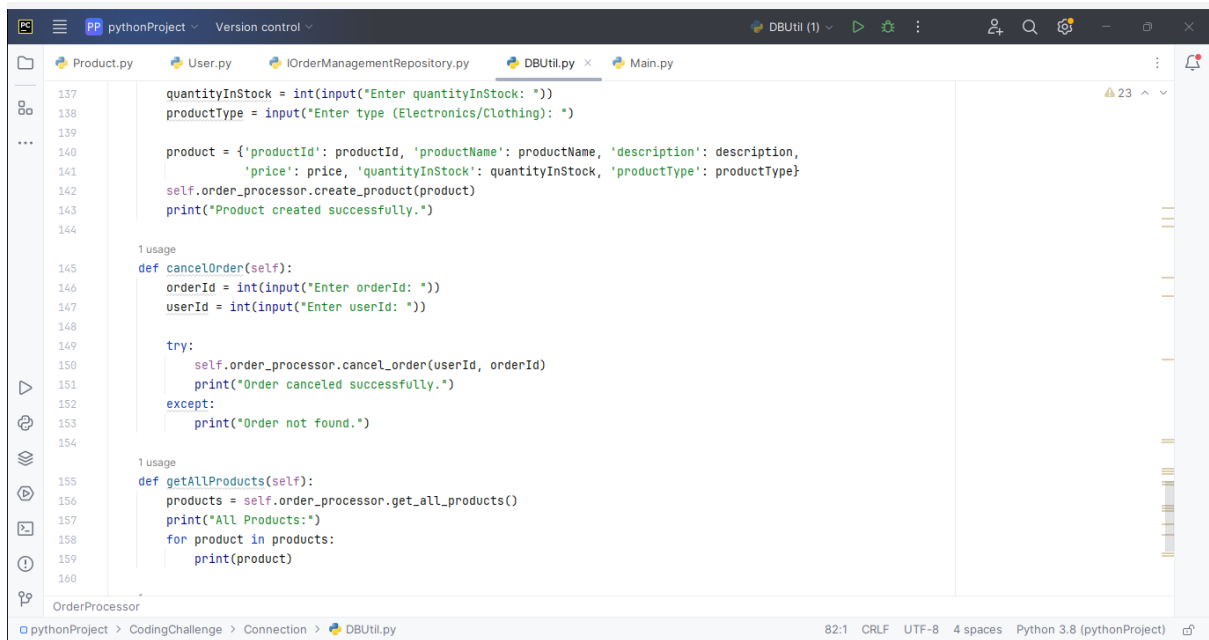
    def createProduct(self):
        productId = int(input("Enter productId: "))
        productName = input("Enter productName: ")
        description = input("Enter description: ")
        price = float(input("Enter price: "))
```

```
pythonProject Version control DBUtil (1)
Product.py User.py IOrderManagementRepository.py DBUtil.py Main.py
self.getAllProducts()
elif choice == '5':
    self.getOrderByUser()
elif choice == '6':
    print("Exiting...")
    break
else:
    print("Invalid choice. Please try again.")

1 usage
def createUser(self):
    userId = int(input("Enter userId: "))
    username = input("Enter username: ")
    password = input("Enter password: ")
    role = input("Enter role (Admin/User): ")

    user = {'userId': userId, 'username': username, 'password': password, 'role': role}
    self.order_processor.create_user(user)
    print("User created successfully.")

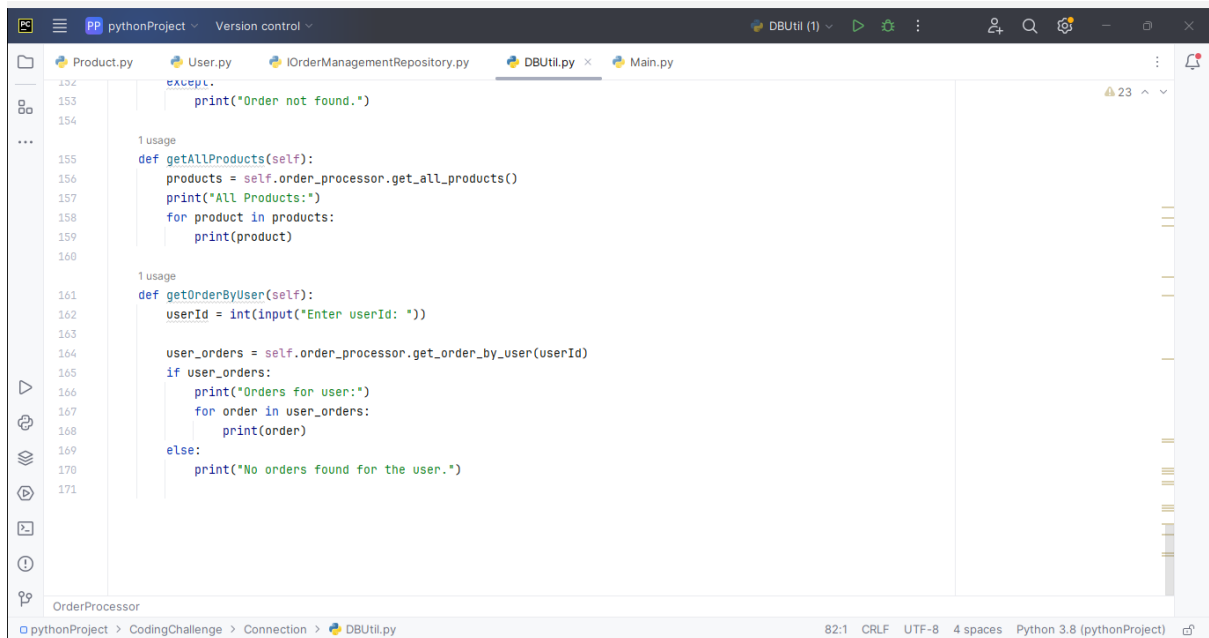
1 usage
def createProduct(self):
    productId = int(input("Enter productId: "))
    productName = input("Enter productName: ")
    description = input("Enter description: ")
    price = float(input("Enter price: "))
```



```
137 quantityInStock = int(input("Enter quantityInStock: "))
138 productType = input("Enter type (Electronics/Clothing): ")
139
140 product = {'productId': productId, 'productName': productName, 'description': description,
141            'price': price, 'quantityInStock': quantityInStock, 'productType': productType}
142 self.order_processor.create_product(product)
143 print("Product created successfully.")
144
145 1 usage
146 def cancelOrder(self):
147     orderId = int(input("Enter orderId: "))
148     userId = int(input("Enter userId: "))
149
150     try:
151         self.order_processor.cancel_order(userId, orderId)
152         print("Order canceled successfully.")
153     except:
154         print("Order not found.")
155
156 1 usage
157 def getAllProducts(self):
158     products = self.order_processor.get_all_products()
159     print("All Products:")
160     for product in products:
161         print(product)
```

OrderProcessor

pythonProject > CodingChallenge > Connection > DBUtil.py 82:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)



```
152 except:
153     print("Order not found.")
154
155 1 usage
156 def getAllProducts(self):
157     products = self.order_processor.get_all_products()
158     print("All Products:")
159     for product in products:
160         print(product)
161
162 1 usage
163 def getOrderByUser(self):
164     userId = int(input("Enter userId: "))
165
166     user_orders = self.order_processor.get_order_by_user(userId)
167     if user_orders:
168         print("Orders for user:")
169         for order in user_orders:
170             print(order)
171     else:
172         print("No orders found for the user.")
```

OrderProcessor

pythonProject > CodingChallenge > Connection > DBUtil.py 82:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject)