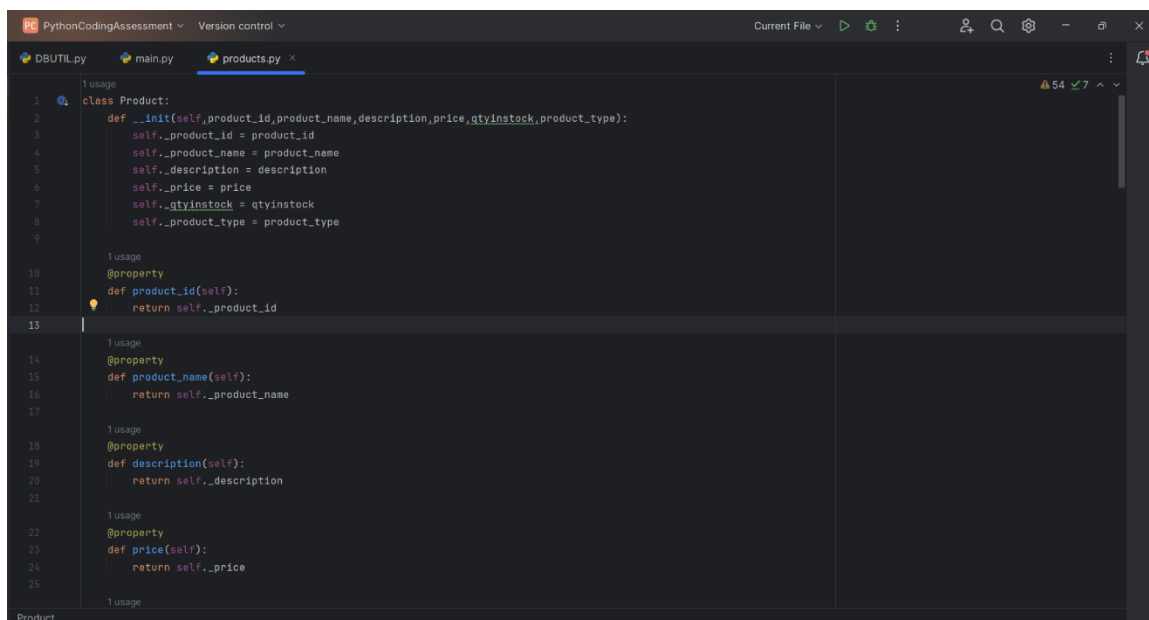


**Name: R. Surya prakash**

## **Coding Challenge – Order Management System**

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.



```
PythonCodingAssessment Version control Current File
DBUTIL.py main.py products.py
1 class Product:
2     def __init__(self, product_id, product_name, description, price, qtyinstock, product_type):
3         self._product_id = product_id
4         self._product_name = product_name
5         self._description = description
6         self._price = price
7         self._qtyinstock = qtyinstock
8         self._product_type = product_type
9
10    1 usage
11    @property
12    def product_id(self):
13        return self._product_id
14
15    1 usage
16    @property
17    def product_name(self):
18        return self._product_name
19
20    1 usage
21    @property
22    def description(self):
23        return self._description
24
25    1 usage
26    @property
27    def price(self):
28        return self._price
29
30    1 usage
Product
```

```
PythonCodingAssessment Version control Current File
DBUTIL.py main.py products.py x

26 1 usage
27 @property
28 def qtyinstock(self):
29     return self._qtyinstock
30
31 1 usage
32 @property
33 def product_type(self):
34     return self._product_type
35
36 1 usage
37 @product_id.setter
38 def product_id(self, pro_id):
39     self.product_id = pro_id
40
41 1 usage
42 @product_name.setter
43 def product_name(self, name):
44     self.product_name = name
45
46 1 usage
47 @description.setter
48 def description(self, desc):
49     self.description = desc
50
51 1 usage
52 @price.setter
53 def price(self, rate):
54     self.price = rate
55
56 1 usage
57 @qtyinstock.setter
58 def qtyinstock(self, quantity):
59     self.qtyinstock = quantity
60
61 1 usage
62 @product_type.setter
63 def product_type(self, pro_type):
64     self.product_type = pro_type
```

3. Create a subclass Electronics that inherits from Product. Add attributes specific to electronics products, such as • brand (String) • warranty Period (int)

```
class Electronics:
    def __init__(self, product_id, product_name, description, price, qtyinstock, product_type, brand, warranty_period):
        super().__init__(product_id, product_name, description, price, qtyinstock, product_type)
        self._brand = brand
        self._warranty_period = warranty_period

    1 usage
    @property
    def brand(self):
        return self._brand

    1 usage
    @property
    def warranty_period(self):
        return self._warranty_period

    @brand.setter
    def brand(self, value):
        self._brand = value

    1 usage
    @warranty_period.setter
    def warranty_period(self, warranty):
        self.warranty_period = warranty
```

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

```

class Clothing(Product):
    def __init__(self, product_id, product_name, description, price, qtyinstock, product_type, size, colour):
        super().__init__(product_id, product_name, description, price, qtyinstock, product_type)
        self._size = size
        self._colour = colour

    1 usage
    @property
    def size(self):
        return self._size

    1 usage
    @property
    def colour(self):
        return self._colour

    1 usage
    @size.setter
    def size(self, size):
        self._size = size

    1 usage
    @colour.setter
    def colour(self, color):
        self._colour = color

```

5. Create a user class with attributes: • userId (int) • username (String) • password (String) • role (String) // "Admin" or "User"

```

class user:
    def __init__(self, user_id, user_name, password, role):
        self._user_id = user_id
        self._user_name = user_name
        self._password = password
        self._role = role

    @property
    def user_id(self):
        return self._user_id

    1 usage
    @property
    def username(self):
        return self._user_name

    1 usage
    @property
    def password(self):
        return self._password

    1 usage
    @property
    def role(self):
        return self._role

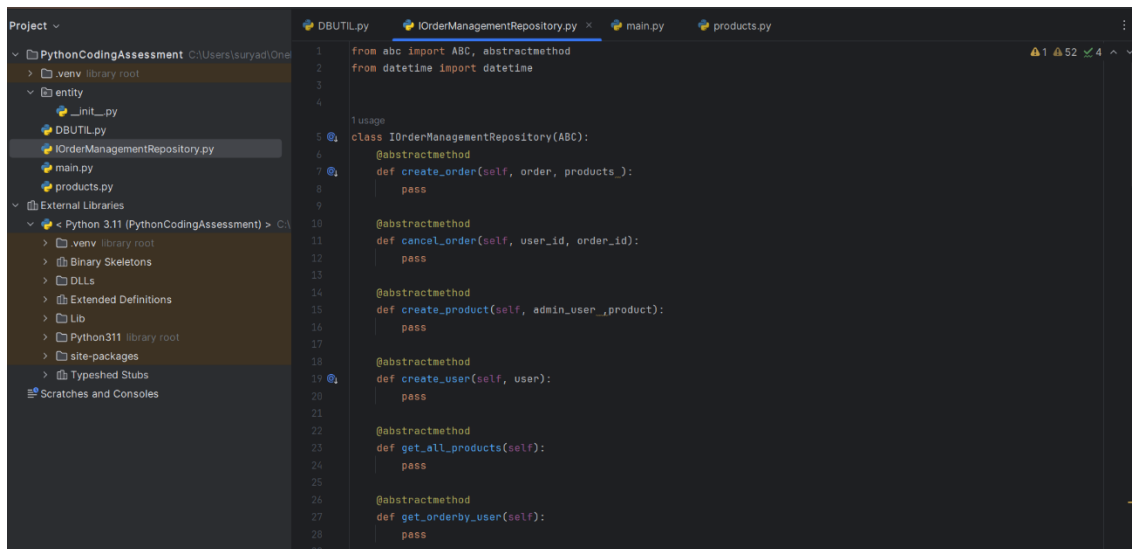
    @username.setter
    def username(self, value):
        self._user_name = value

    1 usage
    @password.setter
    def password(self, val):
        self._password = val

    1 usage
    @role.setter
    def role(self, val):
        self._role = val

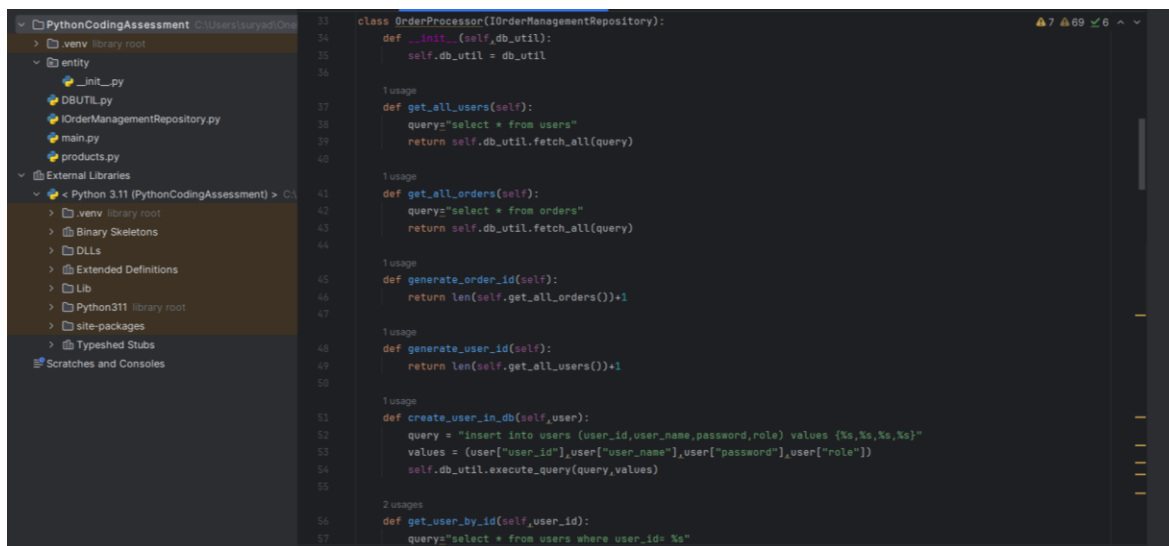
```

6. Define an interface/abstract class named IOrderManagementRepository with methods for: • create Order (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.



```
1 from abc import ABC, abstractmethod
2 from datetime import datetime
3
4
5 class IOrderManagementRepository(ABC):
6     @abstractmethod
7     def create_order(self, order, products):
8         pass
9
10    @abstractmethod
11    def cancel_order(self, user_id, order_id):
12        pass
13
14    @abstractmethod
15    def create_product(self, admin_user, product):
16        pass
17
18    @abstractmethod
19    def create_user(self, user):
20        pass
21
22    @abstractmethod
23    def get_all_products(self):
24        pass
25
26    @abstractmethod
27    def get_orderby_user(self):
28        pass
29
```

7. Implement the IOrderManagementRepository interface/abstract class in a class called Order Processor. This class will be responsible for managing orders.



```
33 class OrderProcessor(IOrderManagementRepository):
34     def __init__(self, db_util):
35         self.db_util = db_util
36
37     1 usage
38     def get_all_users(self):
39         query = "select * from users"
40         return self.db_util.fetch_all(query)
41
42     1 usage
43     def get_all_orders(self):
44         query = "select * from orders"
45         return self.db_util.fetch_all(query)
46
47     1 usage
48     def generate_order_id(self):
49         return len(self.get_all_orders()) + 1
50
51     1 usage
52     def generate_user_id(self):
53         return len(self.get_all_users()) + 1
54
55     1 usage
56     def create_user_in_db(self, user):
57         query = "insert into users (user_id, user_name, password, role) values (%s, %s, %s, %s)"
58         values = (user["user_id"], user["user_name"], user["password"], user["role"])
59         self.db_util.execute_query(query, values)
60
61     2 usages
62     def get_user_by_id(self, user_id):
63         query = "select * from users where user_id= %s"
64
```

This screenshot shows the implementation of database utility methods in `DBUTIL.py`. The file is part of a project named `PythonCodingAssessment`. The methods implemented are:

- `get_user_by_id(self, user_id)`: A method that takes a `user_id` and returns a user object. It uses a query to select from the `users` table where `user_id` is the provided value.
- `get_order_by_id(self, order_id)`: A method that takes an `order_id` and returns an order object. It uses a query to select from the `orders` table where `order_id` is the provided value.
- `create_orderdetail_in_db(self, order_id, product_id, quantity)`: A method that inserts a new order detail into the `order_details` table. It uses a query to insert values for `order_id`, `product_id`, and `quantity`.
- `create_order_in_db(self, order)`: A method that inserts a new order into the `orders` table. It uses a query to insert values for `order_id`, `user_id`, `order_date`, and `totalamt`. It also calls `create_orderdetail_in_db` to insert order details for each product in the order.
- `get_current_datetime(self)`: A method that returns the current datetime.

This screenshot shows the implementation of the `create_order` method in `IOrderManagementRepository.py`. The method takes a `user` and a list of `products` as input. It performs the following steps:

- Check if the user exists. If not, create the user using `create_user`.
- Generate an order ID using `generate_order_id`.
- Calculate the total amount using `sum(product["price"] * product["qty"] for product in products)`.
- Get the current datetime using `get_current_datetime`.
- Create an order object with the following attributes: `order_id`, `user`, `order_date`, `total_amount`, and `products`.
- Insert the order into the database using `create_order_in_db`.
- For each product in the order, insert an order detail into the database using `create_order_detail_in_db`.
- Finally, call `create_order_in_db` again to insert the order details.

This screenshot shows the implementation of the `create_user` and `cancel_order` methods in `IOrderManagementRepository.py`.

- `create_user(self, user)`: A method that takes a `user` object and creates a new user in the database. It generates a user ID, creates the user, and inserts it into the database using `create_user_in_db`.
- `cancel_order(self, user_id, order_id)`: A method that takes a `user_id` and an `order_id` and cancels the order. It checks if the user and order exist. If not, it prints "User or order not found". If they exist, it deletes the order from the `products` table using a query to delete where `product_id` is the provided value.

8. Create DBUtil class and add the following method. • static getDBConn (): Connection Establish a connection to the database and return database Connection

```
1 import mysql.connector
2
3 class DBUTIL:
4     def __init__(self,host,user,password,port,database):
5         self.connections=mysql.connector.connect(
6             host=host,
7             user=user,
8             password=password,
9             port=port,
10            database=database
11        )
12        self.cursor=self.connection.cursor()
13
14    @usage (3 dynamic)
15    def execute_query(self,query,values=None):
16        try:
17            self.cursor.execute(query,values)
18            self.connection.commit()
19        except Exception as e:
20            print(f"Error executing query: {str(e)}")
21            self.connection.rollback()
22
23    @usage (1 dynamic)
24    def fetch_one(self,query,values=None):
25        self.cursor.execute(query,values)
26        return self.cursor.fetchone()
27
28    def close_connection(self):
29        self.cursor.close()
30        self.connection.close()
```

9. Create Order Management 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"

```
126
127
128 def main():
129     db_util = DBUTIL(host="localhost", user="root", password="12345678", port=3306, database="ordermanagement")
130     order_processor = OrderProcessor(db_util)
131
132     while True:
133         print("Menu")
134         print("1. Create user")
135         print("2. create product")
136         print("3. cancel order")
137         print("4. getAllProducts")
138         print("5. getorderbyuser")
139         print("6. exit")
140         option = input("Enter your choice: ")
141
142         if option == "1":
143             username = input("username: ")
144             password = input("password: ")
145             role = input("role (Admin or User): ")
146             user = {"username": username, "password": password, "role": role}
147             OrderProcessor.create_user(user)
148             print("User created successfully.")
149
150         elif option == "2":
151             products = {
152                 "product_name": input("product name: "),
153                 "description": input("Description: "),
154                 "price": float(input("price: ")),
155                 "qtyinstock": input("Quantity in stock :"),
156                 "type": input("product type(Electronic/clothing)"),
157             }
```

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PythonCodingAssessment' with a file structure including 'entity', 'main.py', 'products.py', and 'External Libraries'. The code editor shows a Python script with the following code:

```
158 OrderProcessor.create_product(product)
159 print("Product created successfully.")
160
161 elif option == "3":
162     user_id = int(input("User ID: "))
163     order_id = int(input("Order ID: "))
164     OrderProcessor.cancel_order(user_id, order_id)
165     print("Order cancelled successfully.")
166
167
168 elif option == "4":
169     products = OrderProcessor.get_all_products()
170     print("Products:")
171     for product in products:
172         print(product)
173
174
175 elif option == "5":
176     user_id = int(input("User ID: "))
177     orders = OrderProcessor.get_order_by_user(user_id)
178     for order in orders:
179         print(order)
180
181
182 elif option == "6":
183     print("I QUIT")
184     break
185
186 else:
187     print("valid option")
188
```

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PythonCodingAssessment' with a file structure including 'main.py', 'products.py', and 'External Libraries'. The code editor shows a Python script with the following code:

```
175 elif option == "5":
176     user_id = int(input("User ID: "))
177     orders = OrderProcessor.get_order_by_user(user_id)
178     for order in orders:
179         print(order)
180
181
182 elif option == "6":
183     print("I QUIT")
184     break
185
186 else:
187     print("valid option")
188
189 if __name__ == "__main__":
190     main()

```