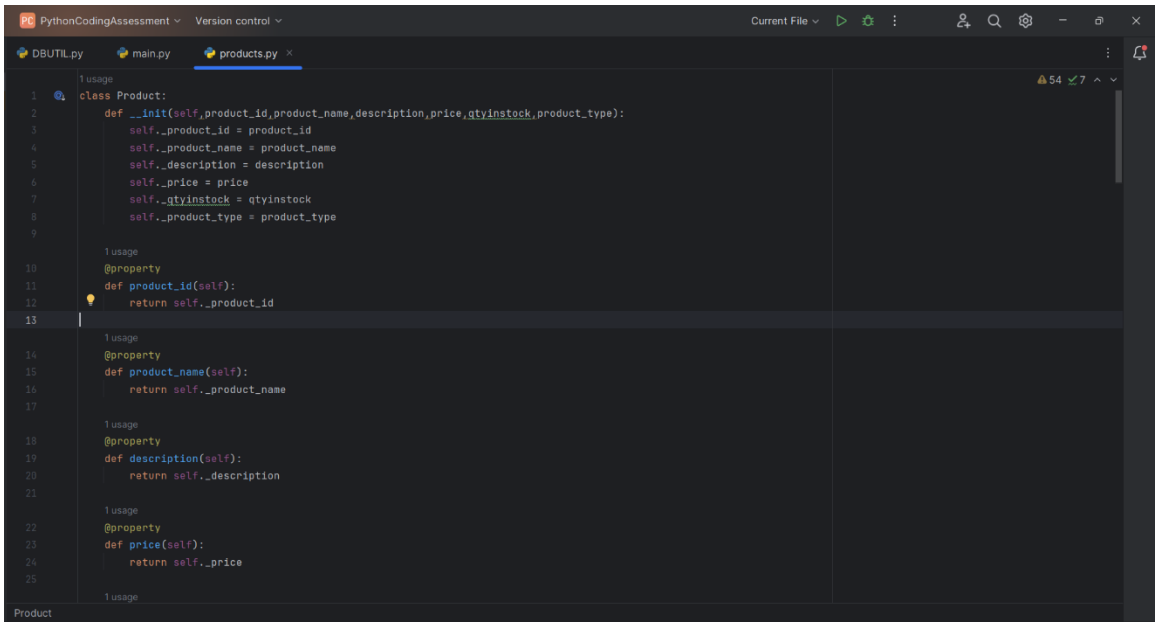
Name: R. Surya prakash

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.



```python
class Product:
    def __init(self,product_id,product_name,description,price,qtyinstock,product_type):
        self._product_id = product_id
        self._product_name = product_name
        self._description = description
        self._price = price
        self._qtyinstock = qtyinstock
        self._product_type = product_type

    @property
    def product_id(self):
        return self._product_id

    @property
    def product_name(self):
        return self._product_name

    @property
    def description(self):
        return self._description

    @property
    def price(self):
        return self._price
```

```python
    1 usage
    @property
    def qtyinstock(self):
        return self._qtyinstock

    1 usage
    @property
    def product_type(self):
        return self._product_type

    1 usage
    @product_id.setter
    def product_id(self,pro_id):
        self.product_id = pro_id

    1 usage
    @product_name.setter
    def product_name(self,name):
        self.product_name = name

    1 usage
    @description.setter
    def description(self,desc):
        self.description = desc

    1 usage
    @price.setter
    def price(self,rate):
        self.price = rate
```

```python
    1 usage
    @qtyinstock.setter
    def qtyinstock(self,quantity):
        self.qtyinstock = quantity

    1 usage
    @product_type.setter
    def product_type(self,pro_type):
        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)

```python
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)

```python
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"

```python
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
```

```python
    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.

```python
from abc import ABC, abstractmethod
from datetime import datetime


class IOrderManagementRepository(ABC):
    @abstractmethod
    def create_order(self, order, products_):
        pass

    @abstractmethod
    def cancel_order(self, user_id, order_id):
        pass

    @abstractmethod
    def create_product(self, admin_user_,product):
        pass

    @abstractmethod
    def create_user(self, user):
        pass

    @abstractmethod
    def get_all_products(self):
        pass

    @abstractmethod
    def get_orderby_user(self):
        pass
```

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

```python
class OrderProcessor(IOrderManagementRepository):
    def __init__(self,db_util):
        self.db_util = db_util

    def get_all_users(self):
        query="select * from users"
        return self.db_util.fetch_all(query)

    def get_all_orders(self):
        query="select * from orders"
        return self.db_util.fetch_all(query)

    def generate_order_id(self):
        return len(self.get_all_orders())+1

    def generate_user_id(self):
        return len(self.get_all_users())+1

    def create_user_in_db(self,user):
        query = "insert into users (user_id,user_name,password,role) values {%s,%s,%s,%s}"
        values = (user["user_id"],user["user_name"],user["password"],user["role"])
        self.db_util.execute_query(query,values)

    def get_user_by_id(self,user_id):
        query="select * from users where user_id= %s"
```

```python
2 usages
def get_user_by_id(self,user_id):
    query="select * from users where user_id= %s"
    values={user_id}
    return self.db_util.fetch_one(query_, values)

1 usage
def get_order_by_id(self,order_id):
    query="select * from users where order_id= %s"
    values={order_id}
    return self.db_util.execute_query(query,values)

1 usage
def create_orderdetail_in_db(self,order_id,product_id,quantity):
    query="insert into order_details (order_id,product_id,quantity) values {%s,%s,%s}"
    values={order_id,product_id,quantity}
    self.db_util.execute_query(query,values)
2 usages
def create_order_in_db(self,order):
    query="insert into orders {order_id,user_id,order_date,totalamt values {%s,%s,%s,%s}"
    values = {order["order_id"],order["user"]["user_id"],order["order_date"],order["totalamt"]}
    self.db_util.execute_query(query,values)

    for product in order["products"]:
        self.create_orderdetail_in_db(order["order_id"],product["product_id"],product["quantity"])

1 usage
def get_current_datetime(self):
    return datetime.now()
```

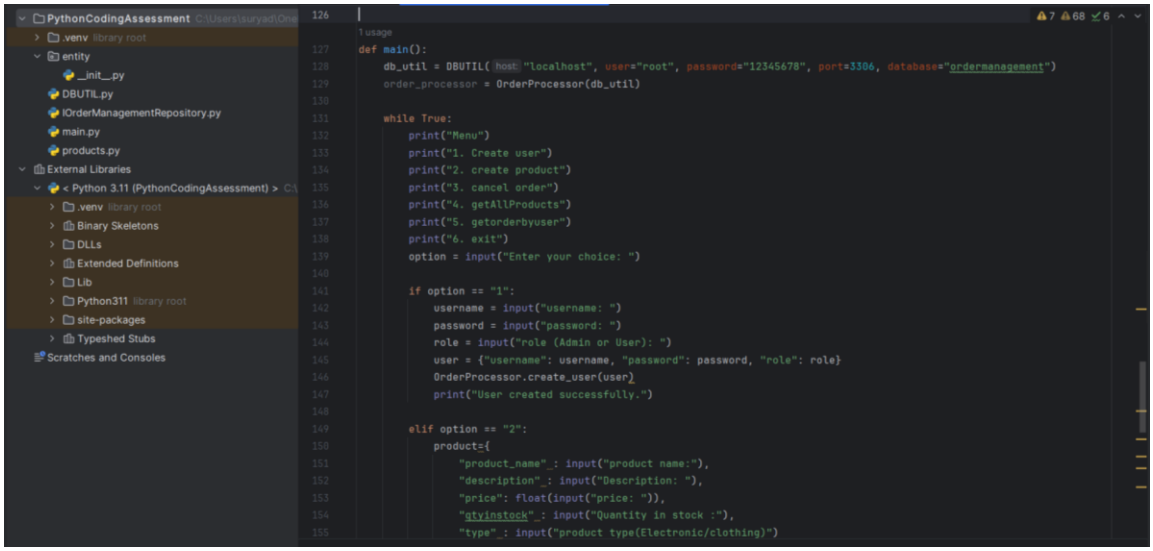
```python
def create_order(self,user,products):
    user_exist=self.get_user_by_id(user["user_id"])

    if user_exist is None:
        self.create_user(user)

    order_id =__self.generate_order_id()

    tot_amt=sum(product["price"]*product["qty"] for product in products)

    order_date=self.get_current_datetime()

    order={
        "order_id"_: order_id,
        "user"_: user,
        "order_date"_: order_date,
        "total_amount"_: tot_amt,
        "products"_: products
    }

    self.create_order_in_db(order)

    for product in products:
        product_id = product["product_id"]
        quantity = product["qtyinstock"]
        self.create_order_detail_in_db(order_id,product_id,quantity)

    self.create_order_in_db(order)
```

```python
2 usages
def create_user(self,user):
    user_id=self.generate_user_id()
    user["user_id"]=user_id
    self.create_user_in_db(user)


1 usage
def cancel_order(self, user_id, order_id):
    user_exist = self.get_user_by_id(user_id)
    order_exist = self.get_order_by_id(order_id)
    if user_exist is None or order_exist is None:
        print("User or order not found")
    else:
        query="delete * from products where product_id = {%s}"
        values={order_id}
        self.db_util.execute_query(query,values)
```

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:
    def __init__(self,host,user,password,port,database):
        self.connection=mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor=self.connection.cursor()

    3 usages (3 dynamic)
    def execute_query(self,query,values=None):
        try:
            self.cursor.execute(query,values)
            self.connection.commit()
        except Exception as e:
            print(f"Error executing query: {str(e)}")
            self.connection.rollback()

    1 usage (1 dynamic)
    def fetch_one(self,query,values=None):
        self.cursor.execute(query,values)
        return self.cursor.fetchone()

    def close_connection(self):
        self.cursor.close()
        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"




PythonCodingAssessment C:\Users\suryad\One
.venv library root
entity
__init__.py
DBUTIL.py
IOrderManagementRepository.py
main.py
products.py
External Libraries
< Python 3.11 (PythonCodingAssessment) > C:\
.venv library root
Binary Skeletons
DLLs
Extended Definitions
Lib
Python311 library root
site-packages
Typeshed Stubs
Scratches and Consoles
OrderProcessor.create_product(product)
print("Product created successfully.")
elif option == "3":
user_id = int(input("User ID: "))
order_id = int(input("Order ID: "))
OrderProcessor.cancel_order(user_id,order_id)
print("Order cancelled successfully.")
elif option == "4":
products = OrderProcessor.get_all_products()
print("Products:")
for product in products:
print(product)
elif option == "5":
user_id = int(input("User ID: "))
orders = OrderProcessor.get_order_by_user(user_id)
for order in orders:
print(order)
elif option == "6":
print("I QUIT")
break
else:
print("valid option")


IOrderManagementRepository.py
main.py
products.py
External Libraries
< Python 3.11 (PythonCodingAssessment) > C:\
.venv library root
Binary Skeletons
DLLs
Extended Definitions
Lib
Python311 library root
site-packages
Typeshed Stubs
Scratches and Consoles
elif option == "5":
user_id = int(input("User ID: "))
orders = OrderProcessor.get_order_by_user(user_id)
for order in orders:
print(order)
elif option == "6":
print("I QUIT")
break
else:
print("valid option")
if __name__ == "__main__":
main()