Coding Challenge: Order Management System

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

• productId (int)

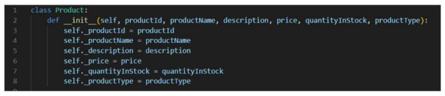
• productName (String)

• description (String)

• price (double)

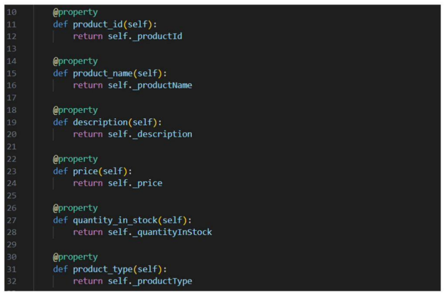
• quantityInStock (int)

• type (String) [Electronics/Clothing]

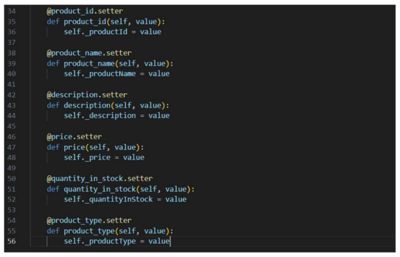


```python
class Product:
    def __init__(self, productId, productName, description, price, quantityInStock, productType):
        self._productId = productId
        self._productName = productName
        self._description = description
        self._price = price
        self._quantityInStock = quantityInStock
        self._productType = productType
```

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



```python
    @property
    def product_id(self):
        return self._productId

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

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

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

    @property
    def quantity_in_stock(self):
        return self._quantityInStock

    @property
    def product_type(self):
        return self._productType
```

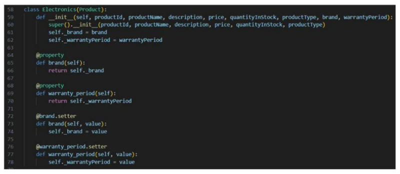


```python
    @product_id.setter
    def product_id(self, value):
        self._productId = value

    @product_name.setter
    def product_name(self, value):
        self._productName = value

    @description.setter
    def description(self, value):
        self._description = value

    @price.setter
    def price(self, value):
        self._price = value

    @quantity_in_stock.setter
    def quantity_in_stock(self, value):
        self._quantityInStock = value

    @product_type.setter
    def product_type(self, value):
        self._productType = value
```

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

• brand (String)

• warrantyPeriod (int)



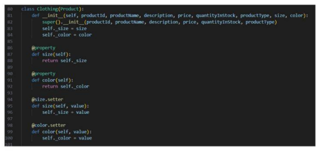
```python
class Electronics(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, productType, brand, warrantyPeriod):
        super().__init__(productId, productName, description, price, quantityInStock, productType)
        self._brand = brand
        self._warrantyPeriod = warrantyPeriod

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

    @property
    def warranty_period(self):
        return self._warrantyPeriod

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

    @warranty_period.setter
    def warranty_period(self, value):
        self._warrantyPeriod = value
```

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

• size (String)

• color (String)

[redacted]



```python
class Clothing(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, productType, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, productType)
        self._size = size
        self._color = color

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

    @property
    def color(self):
        return self._color

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

    @color.setter
    def color(self, value):
        self._color = value
```

5. Create a User class with attributes:

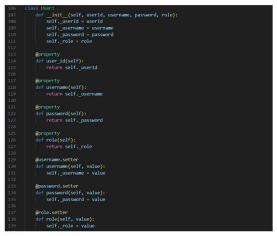
• userId (int)

• username (String)

• password (String)

• role (String) // "Admin" or "User"

[redacted]



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

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

    @property
    def username(self):
        return self._username

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

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

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

    @password.setter
    def password(self, value):
        self._password = value

    @role.setter
    def role(self, value):
        self._role = value
```

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

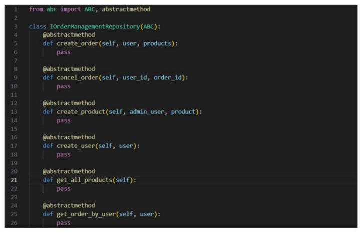
• createProduct(User user, Product product): check the admin user as already present in database and create product and store in database.

• 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.

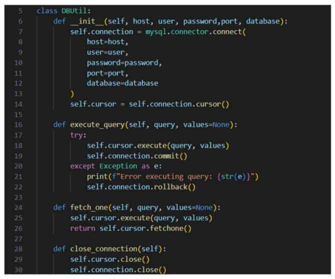
7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.



```python
from abc import ABC, abstractmethod

class IOrderManagementRepository(ABC):
    @abstractmethod
    def create_order(self, user, 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_order_by_user(self, user):
        pass
```

8. Create DBUtil class and add the following method.

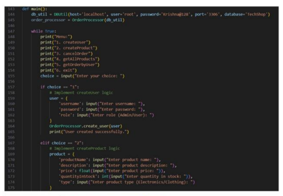
• static getDBConn():Connection Establish a connection to the database and return database Connection

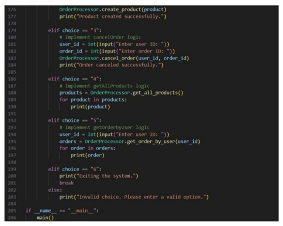


```python
class DBUtil:
    def __init__(self, host, user, password,port, database):
        self.connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor = self.connection.cursor()

    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()

    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()
```

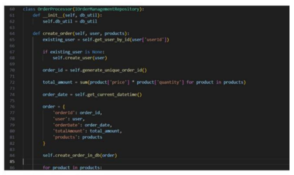
9. Create OrderManagement main class and perform following operation:

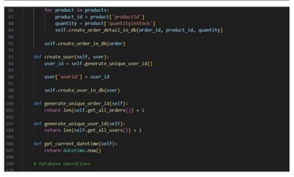
• main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct", "cancelOrder", "getAllProducts", "getOrderbyUser", "exit".





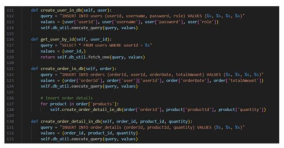
```python
def main():
    db_util = DBUtil(host='localhost', user='root', password='Krishna@128', port='3306', database='TechShop')
    order_processor = OrderProcessor(db_util)

    while True:
        print("Menu:")
        print("1. createUser")
        print("2. createProduct")
        print("3. cancelOrder")
        print("4. getAllProducts")
        print("5. getOrderbyUser")
        print("6. exit")
        choice = input("Enter your choice: ")

        if choice == "1":
            # Implement createUser logic
            user = {
                'username': input("Enter username: "),
                'password': input("Enter password: "),
                'role': input("Enter role (Admin/User): ")
            }
            OrderProcessor.create_user(user)
            print("User created successfully.")

        elif choice == "2":
            # Implement createProduct logic
            product = {
                'productName': input("Enter product name: "),
                'description': input("Enter product description: "),
                'price': float(input("Enter product price: ")),
                'quantityInStock': int(input("Enter quantity in stock: ")),
                'type': input("Enter product type (Electronics/Clothing): ")
            }
            OrderProcessor.create_product(product)
            print("Product created successfully.")

        elif choice == "3":
            # Implement cancelOrder logic
            user_id = int(input("Enter user ID: "))
            order_id = int(input("Enter order ID: "))
            OrderProcessor.cancel_order(user_id, order_id)
            print("Order canceled successfully.")

        elif choice == "4":
            # Implement getAllProducts logic
            products = OrderProcessor.get_all_products()
            for product in products:
                print(product)

        elif choice == "5":
            # Implement getOrderbyUser logic
            user_id = int(input("Enter user ID: "))
            orders = OrderProcessor.get_order_by_user(user_id)
            for order in orders:
                print(order)

        elif choice == "6":
            print("Exiting the system.")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()
```

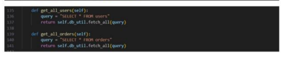




```python
class OrderProcessor(IOrderManagementRepository):
    def __init__(self, db_util):
        self.db_util = db_util

    def create_order(self, user, products):
        existing_user = self.get_user_by_id(user['userId'])

        if existing_user is None:
            self.create_user(user)

        order_id = self.generate_unique_order_id()

        total_amount = sum(product['price'] * product['quantity'] for product in products)

        order_date = self.get_current_datetime()

        order = {
            'orderId': order_id,
            'user': user,
            'orderDate': order_date,
            'totalAmount': total_amount,
            'products': products
        }

        self.create_order_in_db(order)

        for product in products:
```

```python
        for product in products:
            product_id = product['productId']
            quantity = product['quantityInStock']
            self.create_order_detail_in_db(order_id, product_id, quantity)

        self.create_order_in_db(order)

    def create_user(self, user):
        user_id = self.generate_unique_user_id()

        user['userId'] = user_id

        self.create_user_in_db(user)

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

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

    def get_current_datetime(self):
        return datetime.now()

    # Database operations
```





```python
    def create_user_in_db(self, user):
        query = "INSERT INTO users (userId, username, password, role) VALUES (%s, %s, %s, %s)"
        values = (user['userId'], user['username'], user['password'], user['role'])
        self.db_util.execute_query(query, values)

    def get_user_by_id(self, user_id):
        query = "SELECT * FROM users WHERE userId = %s"
        values = (user_id,)
        return self.db_util.fetch_one(query, values)

    def create_order_in_db(self, order):
        query = "INSERT INTO orders (orderId, userId, orderDate, totalAmount) VALUES (%s, %s, %s, %s)"
        values = (order['orderId'], order['user']['userId'], order['orderDate'], order['totalAmount'])
        self.db_util.execute_query(query, values)

        # Insert order details
        for product in order['products']:
            self.create_order_detail_in_db(order['orderId'], product['productId'], product['quantity'])

    def create_order_detail_in_db(self, order_id, product_id, quantity):
        query = "INSERT INTO order_details (orderId, productId, quantity) VALUES (%s, %s, %s)"
        values = (order_id, product_id, quantity)
        self.db_util.execute_query(query, values)
```

```python
    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)
```