Coding Challenge - Order Management System

Shivam Singh

Instructions

- Project submissions should be done through the partcipants' Github repository, and the link should be shared with trainers and Hexavarsity.
- Each section builds upon the previous one, and by the end, you will have a comprehensive **Order Management System** implemented with a strong focus on **SQL**, **control flow statements, loops, arrays, collections, exception handling**, **database interaction**.
- Follow **object-oriented principles** throughout the project. Use classes and objects to model real-world entities, **encapsulate data and behavior**, and **ensure code reusability**.
- Throw **user defined exceptions** from corresponding methods and handled**.**
- The following **Directory structure** is to be followed in the application.
    - **entity/model**
        - Create entity classes in this package. All entity class should not have any business logic.
    - **dao**
        - Create Service Provider interface to showcase functionalities.
        - Create the implementation class for the above interface with db interaction.
    - **exception**
        - Create user defined exceptions in this package and handle exceptions whenever needed.
    - **util**
        - Create a **DBPropertyUtil** class with a static function which takes property file name as parameter and returns connection string.
        - Create a **DBConnUtil** class which holds **static method** which takes connection string as parameter file and returns **connection object(Use method defined in DBPropertyUtil class to get the connection String)**.
- **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.

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

- **brand** (String)
- **warrantyPeriod** (int)

4. Create a subclass **Clothing** that also inherits from **Product**. Add attributes specific to clothing products, such as:
    - **size** (String)
    - **color** (String)
5. Create a **User** class with attributes:
    - **userId** (int)
    - **username** (String)
    - **password** (String)
    - **role** (String) // "Admin" or "User"
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.
8. Create **DBUtil** class and add the following method.
    - **static getDBConn():Connection** Establish a connection to the database and return database Connection
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".

Orders.py

```
from db_connector.db_adapter import get_db_connection


class Order:

    def __init__(self, order_id, user_id, order_date, total_price,
shipping_address):
        self.connection = get_db_connection()
        self.__order_id = order_id
        self.__user_id = user_id
        self.__order_date = order_date
        self.__total_price = total_price
        self.__shipping_address = shipping_address

    def get_order_id(self):
        return self.__order_id

    def get_customer_id(self):
        return self.__user_id

    def get_order_date(self):
        return self.__order_date

    def get_total_price(self):
        return self.__total_price

    def get_shipping_address(self):
        return self.__shipping_address

    def update_order_info(self, user_id=None, order_date=None,
total_price=None, shipping_address=None):
        try:
            my_cursor = self.connection.cursor()

            if user_id:
                sql = '''
                UPDATE Orders SET userId = %s WHERE order_id = %s
                '''
                para = (user_id, self.__order_id)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('User ID updated successfully')

            if order_date:
                sql = '''
                UPDATE Orders SET order_date = %s WHERE order_id = %s
                '''
                para = (order_date, self.__order_id)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Order date updated successfully')

            if total_price:
                sql = '''
                UPDATE Orders SET total_price = %s WHERE order_id = %s
                '''
                para = (total_price, self.__order_id)
```

```
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Total price updated successfully')

            if shipping_address:
                sql = '''
                UPDATE Orders SET shipping_address = %s WHERE order_id = %s
                '''
                para = (shipping_address, self.__order_id)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Shipping address updated successfully')

        except Exception as e:
            print(f'An error occurred: {e}')

    def print_order_info(self):
        print(f'Order ID: {self.__order_id}')
        print(f'User ID: {self.__user_id}')
        print(f'Order Date: {self.__order_date}')
        print(f'Total Price: {self.__total_price}')
        print(f'Shipping Address: {self.__shipping_address}')
```

user.py

```
from db_connector.db_adapter import *


class User:

    def __init__(self, userid, username, password, role):
        self.connection = get_db_connection()
        self.__userId = userid
        self.__user_name = username
        self.__password = password
        self.__role = role

    def get_user_id(self):
        return self.__userId

    def get_user_name(self):
        return self.__user_name

    def get_password(self):
        return self.__password

    def get_user_role(self):
        return self.__role

    def update_user_info(self, username=None, password=None, role=None):
        try:
            my_cursor = self.connection.cursor()

            if username:
                sql = '''
                UPDATE User SET username = %s WHERE userID = %s
                '''
                para = (username, self.__userId)
```

```
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('User Name updated successfully')

            if password:
                sql = '''
                UPDATE User SET password = %s WHERE userID = %s
                '''
                para = (password, self.__userId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('User Password updated successfully')

            if role:
                sql = '''
                UPDATE User SET role = %s WHERE userID = %s
                '''
                para = (role, self.__userId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('User Role updated successfully')

        except Exception as e:
            print(f'An error occurred: {e}')

    def print_user_info(self):
        print('UserID', self.__userId)
        print('User Name', self.__user_name)
        print('Password', self.__password)
        print('Role', self.__role)
```

product.py

```
from db_connector.db_adapter import *


class Product:

    def __init__(self, product_id, product_name, description, price,
quantity_in_stock, product_type):
        self.connection = get_db_connection()
        self.__productId = product_id
        self.__product_name = product_name
        self.__description = description
        self.__price = price
        self.__quantity_in_stock = quantity_in_stock
        self.__type = product_type

    def get_product_name(self):
        return self.__product_name

    def get_description(self):
        return self.__description

    def get_price(self):
        return self.__price

    def get_quantity_in_stock(self):
        return self.__quantity_in_stock
```

```
    def get_product_type(self):
        return self.__type

    def update_product_info(self, product_name=None, description=None,
price=None, quantity_in_stock=None, product_type=None):
        try:
            my_cursor = self.connection.cursor()

            if product_name:
                sql = '''
                UPDATE Product SET productName = %s WHERE productId = %s
                '''
                para = (product_name, self.__productId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Product Name updated successfully')

            if description:
                sql = '''
                UPDATE Product SET description = %s WHERE productId = %s
                '''
                para = (description, self.__productId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Product description updated successfully')

            if price:
                sql = '''
                UPDATE Product SET price = %s WHERE productId = %s
                '''
                para = (price, self.__productId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Product Price updated successfully')

            if quantity_in_stock:
                sql = '''
                UPDATE Product SET quantityInStock = %s WHERE productId = %s
                '''
                para = (quantity_in_stock, self.__productId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Product Quantity_In_Stock updated successfully')

            if product_type:
                sql = '''
                UPDATE Product SET type = %s WHERE productId = %s
                '''
                para = (product_type, self.__productId)
                my_cursor.execute(sql, para)
                self.connection.commit()
                print('Product Type updated successfully')

        except Exception as e:
            print(f'An error occurred: {e}')
```

db_adapter.py

```
import mysql.connector


def get_db_connection():

    config = {
        'user': 'root',
        'password': '765795',
        'host': 'localhost',
        'database': 'omsdb'
    }

    try:
        connection = mysql.connector.connect(**config)

        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None


def get_ids(table_name, id_column_name):
    mydb = get_db_connection()
    my_cursor = mydb.cursor()
    sql = 'SELECT ' + id_column_name + ' FROM ' + table_name + ' ORDER BY ' +
id_column_name + ' DESC LIMIT 1'
    # print(sql)
    my_cursor.execute(sql)
    x = list(my_cursor.fetchone())[0]
    return int(x) + 1
```

customer_exceptions.py

```
class UserNotFound(Exception):
    def __init__(self, message="User not found"):
        self.message = message
        super().__init__(self.message)


class OrderNotFound(Exception):
    def __init__(self, message="Order not found"):
        self.message = message
        super().__init__(self.message)
```

main.py

```
from services.order_manager import OrderManager
from custom_exception.custome_exceptions import UserNotFound, OrderNotFound
class OrderManagement:

    @staticmethod
    def print_menu():
        print("\nOrder Management System Menu:")
        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")

    @staticmethod
    def get_user_input():
        return input("\nEnter your choice (1-6): ")

    @staticmethod
    def create_user(order_manager):
        user_id = input("Enter User ID: ")
        username = input("Enter Username: ")
        password = input("Enter Password: ")
        role = input("Enter Role (Admin/User): ")

        order_manager.create_user(user_id, username, password, role)

    @staticmethod
    def create_product(order_manager):
        admin_user_id = input("Enter Admin User ID: ")
        admin_user = order_manager.get_user_by_id(admin_user_id)

        if admin_user and admin_user.get_user_role() == 'Admin':
            product_name = input("Enter Product Name: ")
            description = input("Enter Product Description: ")
            price = float(input("Enter Product Price: "))
            quantity_in_stock = int(input("Enter Quantity in Stock: "))
            product_type = input("Enter Product Type (Electronics/Clothing): ")

            order_manager.create_product(admin_user, product_name, description,
price, quantity_in_stock, product_type)
        else:
            print("Invalid Admin User ID or User is not an admin.")

    @staticmethod
    def cancel_order(order_manager):
        user_id = input("Enter User ID: ")
        order_id = input("Enter Order ID: ")

        try:
            order_manager.cancel_order(user_id, order_id)
        except OrderNotFound as e:
            print(e)
        except UserNotFound as e:
            print(e)

    @staticmethod
    def get_all_products(order_manager):
```

```
        products = order_manager.get_all_products()
        print("\nAll Products:")
        for product in products:
            print(f"{product.get_product_name()} - {product.get_price()} -
{product.get_quantity_in_stock()} in stock")

    @staticmethod
    def get_order_by_user(order_manager):
        user_id = input("Enter User ID: ")
        orders = order_manager.get_order_by_user(user_id)

        if orders:
            print(f"\nOrders for User ID {user_id}:")
            for order in orders:
                print(f"Order ID: {order.get_order_id()}, Date:
{order.get_order_date()}, Total Price: {order.get_total_price()}")
        else:
            print(f"No orders found for User ID {user_id}")

    @staticmethod
    def main():
        order_manager = OrderManager()

        while True:
            OrderManagement.print_menu()
            choice = OrderManagement.get_user_input()

            if choice == "1":
                OrderManagement.create_user(order_manager)
            elif choice == "2":
                OrderManagement.create_product(order_manager)
            elif choice == "3":
                OrderManagement.cancel_order(order_manager)
            elif choice == "4":
                OrderManagement.get_all_products(order_manager)
            elif choice == "5":
                OrderManagement.get_order_by_user(order_manager)
            elif choice == "6":
                print("Exiting Order Management System.")
                break
            else:
                print("Invalid choice. Please enter a number between 1 and 6.")


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

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| cms                |
| crime              |
| g                  |
| hexabatch2         |
| hmbank             |
| information_schema |
| mysql              |
| omsdb              |
| performance_schema |
| sakila             |
| sis                |
| sys                |
| techshop           |
| techshopdb         |
| world              |
+--------------------+
15 rows in set (0.06 sec)
```

```
mysql> use omsdb;
Database changed
mysql> show tables;
+-------------------+
| Tables_in_omsdb   |
+-------------------+
| orders            |
| product           |
| user              |
+-------------------+
3 rows in set (0.02 sec)
```

```
mysql> select*from orders;
+----------+---------+------------+-------------+------------------+
| order_id | user_id | order_date | total_price | shipping_address |
+----------+---------+------------+-------------+------------------+
|        1 |       1 | 2024-02-06 |        1000 | BOKARO           |
|        2 |       2 | 2024-02-06 |         100 | RANCHI           |
|        3 |       3 | 2024-02-06 |        1000 | CHENNAI          |
|        4 |       4 | 2024-02-06 |         100 | DELHI            |
+----------+---------+------------+-------------+------------------+
4 rows in set (0.00 sec)
```

```
mysql> select*from product;
+------------+------------------+--------------------------+-------+------------------+--------------+
| PRODUCT_ID | PRODUCT_NAME     | DESCRIPTION              | PRICE | QUANTITY_IN_STOCK | PRODUCT_TYPE |
+------------+------------------+--------------------------+-------+------------------+--------------+
|          1 | SCREEN GUARD     | 11D                      |   100 |               50 | Electronics  |
|          2 | TSHIRT           | COTTON                   |   100 |               30 | CLOTHING     |
|          3 | Shoes            | Comfortable running shoes |  1000 |              100 | CLOTHING     |
|          4 | Bluetooth Speaker | Portable wireless speaker |   100 |               20 | Electronics  |
+------------+------------------+--------------------------+-------+------------------+--------------+
4 rows in set (0.01 sec)
```

```
mysql> select*from user;
+--------+----------+---------------------+--------+
| USERID | USERNAME | PASSWORD            | ROLE   |
+--------+----------+---------------------+--------+
|      1 | A        | hashed_password_1   | Admin  |
|      2 | B        | hashed_password_2   | User   |
|      3 | C        | hashed_password_3   | User   |
|      4 | D        | hashed_password_4   | Admin  |
|      5 | abc      | abc                 | user   |
+--------+----------+---------------------+--------+
5 rows in set (0.00 sec)
```

After we run main.py

```
D:\apps\python\python.exe D:\apps\charm\order_management_system\main.py

Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Order by User
6. Exit

Enter your choice (1-6):
```

1.

```
Enter your choice (1-6): 1
Enter User ID: 6
Enter Username: shivam singh
Enter Password: abc
Enter Role (Admin/User): admin
User Created successfully
```

```
mysql> select*from user;
+--------+--------------+--------------------+-------+
| USERID | USERNAME     | PASSWORD           | ROLE  |
+--------+--------------+--------------------+-------+
|      1 | A            | hashed_password_1  | Admin |
|      2 | B            | hashed_password_2  | User  |
|      3 | C            | hashed_password_3  | User  |
|      4 | D            | hashed_password_4  | Admin |
|      5 | abc          | abc                | user  |
|      6 | shivam singh | abc                | admin |
+--------+--------------+--------------------+-------+
6 rows in set (0.00 sec)
```

2.

```
Enter your choice (1-6): 2
Enter Admin User ID: 1
Enter Product Name: Samsung Galaxy S22 Ultra
Enter Product Description: 1TB
Enter Product Price: 100000
Enter Quantity in Stock: 12
Enter Product Type (Electronics/Clothing): electronics
```

3.

```
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Order by User
6. Exit

Enter your choice (1-6): 3
Enter User ID: 2
Enter Order ID: 1
Order Cancelled successfully
```

4.

```
Enter your choice (1-6): 4

All Products:
SCREEN GUARD - 100 - 50 in stock
TSHIRT - 100 - 30 in stock
Shoes - 1000 - 100 in stock
Bluetooth Speaker - 100 - 20 in stock
```

5.

```
Enter your choice (1-6): 5
Enter User ID: 100

No orders found for User ID 100
```

6.

```
Enter your choice (1-6): 6
Exiting Order Management System.

Process finished with exit code 0
```