CODING CHALLENGE

Order Management System

Agenda:

Implementation of SQL

Implementation of OOPS

BY

JASWANTH KUAMR S

Batch 4

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

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

Code :

Product

```
class Product:
    def __init__(self, product_id, product_name, description, price,
quantity_in_stock, product_type):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
        self.quantity_in_stock = quantity_in_stock
        self.product_type = product_type

    def __str__(self):
        return f"{self.product_name} ({self.product_type}) - ${self.price}"
```

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

Electronics

```
class Electronics:
    def __init__(self, product_id, product_name, description, price,
quantity_in_stock, product_type, brand, warranty_period):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
        self.quantity_in_stock = quantity_in_stock
        self.product_type = product_type
        self.brand = brand
        self.warranty_period = warranty_period
```

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

clothing

```
class Clothing:
    def __init__(self, product_id, product_name, description, price, quantity_in_stock, product_type, size, color):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
        self.quantity_in_stock = quantity_in_stock
        self.product_type = product_type
        self.size = size
        self.color = color
```

5. Create a User class with attributes:

user

```
class User:
    def __init__(self, user_id, username, password, role):
        self.user_id = user_id
        self.username = username
        self.password = password
        self.role = role
```

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

```
CREATE DATABASE IF NOT EXISTS order_management;
USE order_management;

CREATE TABLE IF NOT EXISTS users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL,
    role VARCHAR(20) CHECK (role IN ('Admin', 'User'))
);

CREATE TABLE IF NOT EXISTS products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    description TEXT,
    price DOUBLE,
    quantity_in_stock INT,
    product_type VARCHAR(50)
);

CREATE TABLE IF NOT EXISTS electronics (
    product_id INT PRIMARY KEY,
    brand VARCHAR(50),
    warranty_period INT,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

CREATE TABLE IF NOT EXISTS clothing (
    product_id INT PRIMARY KEY,
    size VARCHAR(10),
    color VARCHAR(30),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
```

```
);

CREATE TABLE IF NOT EXISTS orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE IF NOT EXISTS order_items (
    order_id INT,
    product_id INT,
    quantity INT,
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    PRIMARY KEY (order_id, product_id)
);
```

7. Define an interface/abstract class named OrderManagementRepository with methods

Implement the OrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.[DAO]

OrderManagementRepository

```
from abc import ABC, abstractmethod

class OrderManagementRepository(ABC):

    @abstractmethod
    def create_user(self, user):
        pass

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

    @abstractmethod
    def create_order(self, user, product_list):
        pass

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

    @abstractmethod
    def get_all_products(self):
        pass

    @abstractmethod
    def get_order_by_user(self, user):
        pass
```

OrderProcessor

```
import mysql.connector
from dao.OrderManagementRepository import OrderManagementRepository
from util.DBConnUtil import DBConnUtil
from exception.UserNotFoundException import UserNotFoundException
from exception.OrderNotFoundException import OrderNotFoundException

class OrderProcessor(OrderManagementRepository):

    def __init__(self):
        self.conn = DBConnUtil.get_connection()
        self.cursor = self.conn.cursor(dictionary=True)

    def create_user(self, user):
        self.cursor.execute("SELECT * FROM users WHERE user_id = %s",
(user.user_id,))
        if not self.cursor.fetchone():
            self.cursor.execute(
                "INSERT INTO users (user_id, username, password, role) VALUES
(%s, %s, %s, %s)",
                (user.user_id, user.username, user.password, user.role)
            )
            self.conn.commit()

    def create_product(self, user, product):
        self.cursor.execute("SELECT * FROM users WHERE user_id = %s AND
role = 'Admin'", (user.user_id,))
        if not self.cursor.fetchone():
            raise UserNotFoundException("Admin user not found.")

        self.cursor.execute("SELECT * FROM products WHERE product_id =
%s", (product.product_id,))
        if self.cursor.fetchone():
            print("Product already exists.")
            return

        self.cursor.execute(
            "INSERT INTO products (product_id, product_name, description, price,
quantity_in_stock, product_type) "
            "VALUES (%s, %s, %s, %s, %s, %s)",
            (product.product_id, product.product_name, product.description,
             product.price, product.quantity_in_stock, product.product_type)
```

```
        )

        if product.product_type == "Electronics":
            self.cursor.execute(
                "INSERT INTO electronics (product_id, brand, warranty_period)
VALUES (%s, %s, %s)",
                (product.product_id, product.brand, product.warranty_period)
            )
        elif product.product_type == "Clothing":
            self.cursor.execute(
                "INSERT INTO clothing (product_id, size, color) VALUES (%s, %s,
%s)",
                (product.product_id, product.size, product.color)
            )

        self.conn.commit()

    def create_order(self, user, product_list):
        self.cursor.execute("SELECT * FROM users WHERE user_id = %s",
(user.user_id,))
        if not self.cursor.fetchone():
            self.create_user(user)

        self.cursor.execute("INSERT INTO orders (user_id) VALUES (%s)",
(user.user_id,))
        order_id = self.cursor.lastrowid

        for item in product_list:
            self.cursor.execute(
                "INSERT INTO order_items (order_id, product_id, quantity)
VALUES (%s, %s, %s)",
                (order_id, item['product_id'], item['quantity'])
            )

        self.conn.commit()
        print(f"Order #{order_id} created for user {user.user_id}")

    def cancel_order(self, user_id, order_id):
        self.cursor.execute("SELECT * FROM users WHERE user_id = %s",
(user_id,))
        if not self.cursor.fetchone():
            raise UserNotFoundException()
```

```
        self.cursor.execute("SELECT * FROM orders WHERE order_id = %s
AND user_id = %s", (order_id, user_id))
        if not self.cursor.fetchone():
            raise OrderNotFoundException()

        self.cursor.execute("DELETE FROM order_items WHERE order_id =
%s", (order_id,))
        self.cursor.execute("DELETE FROM orders WHERE order_id = %s",
(order_id,))
        self.conn.commit()

    def get_all_products(self):
        self.cursor.execute("SELECT * FROM products")
        return self.cursor.fetchall()

    def get_order_by_user(self, user):
        self.cursor.execute("SELECT * FROM orders WHERE user_id = %s",
(user.user_id,))
        orders = self.cursor.fetchall()
        result = []
        for order in orders:
            self.cursor.execute("SELECT * FROM order_items WHERE order_id =
%s", (order['order_id'],))
            items = self.cursor.fetchall()
            result.append({'order_id': order['order_id'], 'items': items})
        return result
```

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

```
import os
import configparser

class DBPropertyUtil:
    @staticmethod
    def get_connection_string(filename):
        config = configparser.ConfigParser()
        full_path = os.path.join(os.path.dirname(__file__), '..', filename)
        config.read(full_path)

        if not config.has_section("mysql"):
            raise Exception("Missing [mysql] section in db.properties")

        return {
            "host": config.get("mysql", "host"),
            "user": config.get("mysql", "user"),
            "password": config.get("mysql", "password"),
            "database": config.get("mysql", "database")
        }


import mysql.connector
from util.DBPropertyUtil import DBPropertyUtil

class DBConnUtil:
    @staticmethod
    def get_connection():
        props = DBPropertyUtil.get_connection_string("db.properties")
        conn = mysql.connector.connect(
            host=props["host"],
            user=props["user"],
            password=props["password"],
            database=props["database"]
        )
        return conn
```

9. Create OrderManagement main class and perform following operation: • main method . Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct", "cancelOrder", "getAllProducts", "getOrderbyUser", "exit".

```
from dao.OrderProcessor import OrderProcessor
from entity.user import User
from entity.electronics import Electronics
from entity.clothing import Clothing

def main():
    processor = OrderProcessor()

    while True:
        print("\n===== Order Management System =====")
        print("1. Create User")
        print("2. Create Product")
        print("3. Create Order")
        print("4. Cancel Order")
        print("5. Get All Products")
        print("6. Get Orders by User")
        print("7. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            uid = int(input("User ID: "))
            uname = input("Username: ")
            pwd = input("Password: ")
            role = input("Role (Admin/User): ")
            user = User(uid, uname, pwd, role)
            processor.create_user(user)
            print("User created.")

        elif choice == '2':
            uid = int(input("Admin User ID: "))
            user = User(uid, "", "", "Admin")

            pid = int(input("Product ID: "))
            name = input("Product Name: ")
            desc = input("Description: ")
            price = float(input("Price: "))
            qty = int(input("Quantity: "))
```

```
            ptype = input("Type (Electronics/Clothing): ")

            if ptype == "Electronics":
                brand = input("Brand: ")
                warranty = int(input("Warranty (months): "))
                product = Electronics(pid, name, desc, price, qty, ptype, brand,
warranty)
            elif ptype == "Clothing":
                size = input("Size: ")
                color = input("Color: ")
                product = Clothing(pid, name, desc, price, qty, ptype, size, color)
            else:
                print("Invalid product type!")
                continue

            processor.create_product(user, product)
            print("Product created.")

        elif choice == '3':
            uid = int(input("User ID: "))
            uname = input("Username: ")
            pwd = input("Password: ")
            role = input("Role: ")
            user = User(uid, uname, pwd, role)

            num_items = int(input("Number of products in the order: "))
            products = []
            for _ in range(num_items):
                pid = int(input("Product ID: "))
                qty = int(input("Quantity: "))
                products.append({"product_id": pid, "quantity": qty})

            processor.create_order(user, products)

        elif choice == '4':
            uid = int(input("User ID: "))
            oid = int(input("Order ID: "))
            try:
                processor.cancel_order(uid, oid)
                print("Order cancelled.")
            except Exception as e:
                print(e)
```

```
        elif choice == '5':
            products = processor.get_all_products()
            for p in products:
                print(p)

        elif choice == '6':
            uid = int(input("User ID: "))
            uname = input("Username: ")
            pwd = input("Password: ")
            role = input("Role: ")
            user = User(uid, uname, pwd, role)

            orders = processor.get_order_by_user(user)
            for order in orders:
                print(f"Order ID: {order['order_id']}")
                for item in order['items']:
                    print(f"  Product ID: {item['product_id']}, Quantity:
{item['quantity']}")

        elif choice == '7':
            print("Exiting Order Management System.")
            break
        else:
            print("Invalid choice. Try again.")

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

IMPLEMENTATION OF OPERATIONS:

1.create user /admin

User creation :

```
Enter your choice: 1
User ID: 1
Username: jash
Password: jash
Role (Admin/User): user
User created.
```

Admin creation :

```
Enter your choice: 1
User ID: 2
Username: vara
Password: jash
Role (Admin/User): admin
User created.
```

Output from the database:

user_id	username	password	role
1	jash	jash	user
2	vara	jash	admin
NULL	NULL	NULL	NULL

2.create product

```
Enter your choice: 2
Admin User ID: 2
Product ID: 1
Product Name: iphone 16 pro max
Description: flagship
Price: 150000
Quantity: 50
Type (Electronics/Clothing): Electronics
Brand: iphone
Warranty (months): 2
Product created.
```

Output from the database:

product_id	product_name	description	price	quantity_in_stock	product_type
1	iphone 16 pro max	flagship	150000	50	Electronics
NULL	NULL	NULL	NULL	NULL	NULL

3.create order

```
Enter your choice: 3
User ID: 1
Username: jash
Password: jash
Role: user
Number of products in the order: 1
Product ID: 1
Quantity: 5
Order #3 created for user 1
```

Output from the database:

order_id	user_id	order_date
3	1	2025-04-09 21:36:03
NULL	NULL	NULL

4.Cancel order:

```
Enter your choice: 4
User ID: 1
Order ID: 3
Order cancelled.
```

Output from the database:

order_id	user_id	order_date
NULL	NULL	NULL

5. get all products

```
1. Create User
2. Create Product
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders by User
7. Exit
Enter your choice: 5
{'product_id': 1, 'product_name': 'iphone 16 pro max', 'description': 'flagship', 'price': 150000.0, 'quantity_in_stock': 50,
```

6. getOrderbyUser

```
Enter your choice: 6
User ID: 1
Username: jash
Password: jash
Role: user
Order ID: 4
  Product ID: 1, Quantity: 10
```

End of the document