CODING CHALLENGE:
ORDER MANAGEMENT

Ahmed Sherif

Objective:

Develop a comprehensive Order Management System focusing on database interactions, object-oriented principles, and exception handling.

Key Components

1. **Class Structure**:
    - Base Product class with electronics/clothing subclasses
    - User class with admin/user roles
    - OrderProcessor implementing OrderManagementRepository interf ace
2. **Database Integration**:
    - Create SQL schema mirroring class attributes
    - Implement DBPropertyUtil and DBConnUtil for connection management
    - Use prepared statements for all DB operations
3. **Core Functionalities**:
    - User registration and authentication
    - Product creation (admin-only)
    - Order creation/cancellation
    - Product/order retrieval
4. **Exception Handling**:
    - Custom exceptions (UserNotFound, OrderNotFound)
    - Proper exception propagation and handling
5. **Technical Requirements**:
    - Strict directory structure (entity, dao, exception, util, main packages)
    - Menu-driven main application

1) Entities:

Entity/Products.py

```
class Product:
    def __init__(self, product_name, description, price, quantity_in_stock, product_type):
        self._product_name = product_name
        self._description = description
        self._price = price
        self._quantity_in_stock = quantity_in_stock
        self._type = product_type


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

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

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

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

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

    @price.setter
    def price(self, value):
        if value >= 0:
            self._price = value
        else:
            raise ValueError("Price cannot be negative")

    @property
    def quantity_in_stock(self):
        return self._quantity_in_stock
```

```
    @quantity_in_stock.setter
    def quantity_in_stock(self, value):
        if value >= 0:
            self._quantity_in_stock = value
        else:
            raise ValueError("Quantity cannot be negative")

    @property
    def type(self):
        return self._type

    @type.setter
    def type(self, value):
        if value in ["Electronics", "Clothing"]:
            self._type = value
        else:
            raise ValueError("Product type must be either 'Electronics' or 'Clothing'")
```

Entity/Users.py

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

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

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

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

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

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

```
    @role.setter
    def role(self, value):
        if value in ["Admin", "User"]:
            self._role = value
        else:
            raise ValueError("Role must be either 'Admin' or 'User'")

    def __str__(self):
        return f"User ID: {self.user_id}, Username: {self.username}, Role: {self.role}"
```

Entity/Clothing.py

```
from entity.product import Product

class Clothing(Product):
    def __init__(self, product_id, product_name, description, price, quantity_in_stock, size, color):
        super().__init__(product_id, product_name, description, price, quantity_in_stock, "Clothing")
        self._size = size
        self._color = color

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

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

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

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

    def __str__(self):
        return super().__str__() + f", Size: {self.size}, Color: {self.color}"
```

Entity/Electronics

```
from entity.product import Product
class Electronics(Product):
    def __init__(self, product_id, product_name, description, price, quantity_in_stock, brand,
warranty_period):
        super().__init__(product_id, product_name, description, price, quantity_in_stock, 'Electronics')
        self.brand = brand
        self.warranty_period = warranty_period

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

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

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

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

    def __str__(self):
        return (super().__str__() +
                f", Brand: {self.brand}, Warranty: {self.warranty_period} months")
```

2) DAO

Dao/order_management_repository.py

```
from abc import ABC, abstractmethod
from typing import List, Dict, Any
from entity.user import User
from entity.product import Product
from exception.ordernotfound import OrderNotFoundException
from exception.usernotfoundexception import UserNotFoundException

class OrderManagementRepository(ABC):
    @abstractmethod
```

```
    def create_user(self, user: User) -> bool:
        pass

    @abstractmethod
    def create_product(self, user: User, product: Product) -> bool:
        pass

    @abstractmethod
    def create_order(self, user: User, products: List[Product]) -> bool:
        pass

    @abstractmethod
    def cancel_order(self, user_id: int, order_id: int) -> bool:
        pass

    @abstractmethod
    def get_all_products(self) -> List[Dict[str, Any]]:
        pass

    @abstractmethod
    def get_order_by_user(self, user_id: int) -> List[Dict[str, Any]]:
        pass
```

Dao.order_processor.py

```
import mysql.connector
from mysql.connector import Error
from typing import List, Dict, Any
from dao.order_management_repository import OrderManagementRepository
from entity.user import User
from entity.product import Product
from entity.electronics import Electronics
from entity.clothing import Clothing
from exception.usernotfoundexception import UserNotFoundException
from exception.ordernotfound import OrderNotFoundException
from exception.productnotfoundexception import ProductNotFoundException
from util.db_util import connect_db

class OrderProcessor(OrderManagementRepository):
    def __init__(self):
        self.connection = None
        try:
            self.connection = connect_db()
            if not self.connection or not self.connection.is_connected():
                raise Exception("failed to establish database connection")
```

```
        except Exception as e:
            print(f"initialization error: {e}")
            if self.connection:
                self.connection.close()
            raise

    def __del__(self):
        if hasattr(self, 'connection') and self.connection and self.connection.is_connected():
            self.connection.close()

    def _user_exists(self, user_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "select userid from users where userid = %s"
            cursor.execute(query, (user_id,))
            return cursor.fetchone() is not None
        except Error as e:
            print(f"error checking user existence: {e}")
            return False
        finally:
            if cursor:
                cursor.close()

    def _product_exists(self, product_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "select productid from products where productid = %s"
            cursor.execute(query, (product_id,))
            return cursor.fetchone() is not None
        except Error as e:
            print(f"error checking product existence: {e}")
            return False
        finally:
            if cursor:
                cursor.close()

    def _order_exists(self, order_id: int) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "select orderid from orders where orderid = %s"
            cursor.execute(query, (order_id,))
            return cursor.fetchone() is not None
        except Error as e:
            print(f"error checking order existence: {e}")
```

```
            return False
        finally:
            if cursor:
                cursor.close()

    def create_user(self, username: str, password: str, role: str) -> int:
        try:
            cursor = self.connection.cursor()
            query = "insert into users (username, password, role) values (%s, %s, %s)"
            cursor.execute(query, (username, password, role))
            self.connection.commit()
            return cursor.lastrowid
        except Error as e:
            print(f"error creating user: {e}")
            self.connection.rollback()
            return -1
        finally:
            if cursor:
                cursor.close()

    def create_product(self, admin_user_id: int, product_data: Dict[str, Any]) -> int:
        try:
            cursor = self.connection.cursor()
            user_check_query = "select userid from users where userid = %s and role = 'admin'"
            cursor.execute(user_check_query, (admin_user_id,))
            admin_user = cursor.fetchone()

            if not admin_user:
                raise UserNotFoundException("admin user not found or doesn't have privileges")

            if product_data['type'].lower() == 'electronics':
                query = """insert into products
                        (productname, description, price, quantityinstock, type, brand, warrantyperiod)
                        values (%s, %s, %s, %s, %s, %s, %s)"""
                values = (
                    product_data['product_name'],
                    product_data['description'],
                    product_data['price'],
                    product_data['quantity_in_stock'],
                    'Electronics',
                    product_data['brand'],
                    product_data['warranty_period']
                )
            elif product_data['type'].lower() == 'clothing':
```

```
            query = """insert into products
                    (productname, description, price, quantityinstock, type, size, color)
                    values (%s, %s, %s, %s, %s, %s, %s)"""
            values = (
                product_data['product_name'],
                product_data['description'],
                product_data['price'],
                product_data['quantity_in_stock'],
                'Clothing',
                product_data['size'],
                product_data['color']
            )
        else:
            raise ValueError("invalid product type")

        cursor.execute(query, values)
        self.connection.commit()
        return cursor.lastrowid
    except Error as e:
        print(f"error creating product: {e}")
        self.connection.rollback()
        return -1
    finally:
        if cursor:
            cursor.close()

def create_order(self, user_id: int, product_ids: List[int]) -> bool:
    if not self._user_exists(user_id):
        raise UserNotFoundException()

    try:
        cursor = self.connection.cursor()
        order_query = "insert into orders (userid) values (%s)"
        cursor.execute(order_query, (user_id,))
        order_id = cursor.lastrowid

        for product_id in product_ids:
            if not self._product_exists(product_id):
                raise ProductNotFoundException(f"product id {product_id} not found")

            detail_query = "insert into orderdetails (orderid, productid, quantity) values (%s, %s, %s)"
            cursor.execute(detail_query, (order_id, product_id, 1))

            update_query = "update products set quantityinstock = quantityinstock - 1 where productid =
```

```
%s"
                cursor.execute(update_query, (product_id,))

            self.connection.commit()
            return True
        except Error as e:
            print(f"error creating order: {e}")
            self.connection.rollback()
            return False
        finally:
            if cursor:
                cursor.close()

    def cancel_order(self, user_id: int, order_id: int) -> bool:
        if not self._user_exists(user_id):
            raise UserNotFoundException()
        if not self._order_exists(order_id):
            raise OrderNotFoundException()

        try:
            cursor = self.connection.cursor()
            get_products_query = "select productid from orderdetails where orderid = %s"
            cursor.execute(get_products_query, (order_id,))
            product_ids = [row[0] for row in cursor.fetchall()]

            for pid in product_ids:
                update_query = "update products set quantityinstock = quantityinstock + 1 where productid =
%s"
                cursor.execute(update_query, (pid,))

            delete_details_query = "delete from orderdetails where orderid = %s"
            cursor.execute(delete_details_query, (order_id,))

            delete_order_query = "delete from orders where orderid = %s"
            cursor.execute(delete_order_query, (order_id,))

            self.connection.commit()
            return cursor.rowcount > 0
        except Error as e:
            print(f"error canceling order: {e}")
            self.connection.rollback()
            return False
        finally:
            if cursor:
```

```
            cursor.close()

    def get_all_products(self) -> List[Dict[str, Any]]:
        try:
            cursor = self.connection.cursor(dictionary=True)
            query = "select * from products"


            cursor.execute(query)
            return cursor.fetchall()
        except Error as e:
            print(f"error fetching products: {e}")
            return []
        finally:
            if cursor:
                cursor.close()

    def get_order_by_user(self, user_id: int) -> List[Dict[str, Any]]:
        if not self._user_exists(user_id):
            raise UserNotFoundException()

        try:
            cursor = self.connection.cursor(dictionary=True)
            query = """select p.* from products p
                    join orderdetails od on p.productid = od.productid
                    join orders o on od.orderid = o.orderid
                    where o.userid = %s"""


            cursor.execute(query, (user_id,))
            return cursor.fetchall()
        except Error as e:
            print(f"error fetching user orders: {e}")
            return []


        finally:
            if cursor:
                cursor.close()
```

3) Exceptions:

Exceptions/OrderNotFoundException.py

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

```
        self.message = message
        super().__init__(self.message)
```

Exceptions/UserNotFoundException.py

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

Exceptions/ProdcutNotFoundException.py

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

4) UTIL

Util/db_util.py

```
import mysql.connector
from mysql.connector import Error

def connect_db():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="ordermanagement",
            auth_plugin='mysql_native_password'
        )
        print("Database connection successful")
        return connection
    except Error as e:
        print(f"Error connecting to MySQL: {e}")
        return None
```

db.properties

```
[database]
host=localhost
port=3306
name=ordermanagement
user=root
password=root
```

5) MAIN

Main/main.py

```
from dao.order_processor import OrderProcessor
from exception.usernotfoundexception import UserNotFoundException
from exception.ordernotfound import OrderNotFoundException


class MainModule:
    def __init__(self):
        self.order_processor = OrderProcessor()

    def display_menu(self):
        print("\nOrder Management System")
        print("1. Create User")
        print("2. Create Product (Admin only)")
        print("3. Create Order")
        print("4. Cancel Order")
        print("5. Get All Products")
        print("6. Get Orders by User")
        print("7. Exit")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")

            try:
                if choice == "1":
                    self.create_user()
                elif choice == "2":
                    self.create_product()
                elif choice == "3":
                    self.create_order()
                elif choice == "4":
                    self.cancel_order()
                elif choice == "5":
                    self.get_all_products()
                elif choice == "6":
                    self.get_orders_by_user()
                elif choice == "7":
                    print("Exiting the system...")
                    break
```

```
            else:
                print("Invalid choice. Please try again.")
        except Exception as e:
            print(f"Error: {e}")

def create_user(self):
    print("\nCreate New User")
    username = input("Enter username: ")
    password = input("Enter password: ")
    role = input("Enter role (Admin/User): ").capitalize()

    user_id = self.order_processor.create_user(username, password, role)
    if user_id != -1:
        print(f"User created successfully with ID: {user_id}")
    else:
        print("Failed to create user")

def create_product(self):
    print("\nCreate New Product (Admin Only)")
    admin_id = int(input("Enter your admin user ID: "))

    product_type = input("Enter product type (Electronics/Clothing): ").lower()
    product_data = {
        'product_name': input("Enter product name: "),
        'description': input("Enter description: "),
        'price': float(input("Enter price: ")),
        'quantity_in_stock': int(input("Enter quantity in stock: ")),
        'type': product_type
    }

    if product_type == "electronics":
        product_data['brand'] = input("Enter brand: ")
        product_data['warranty_period'] = int(input("Enter warranty period (months): "))
    elif product_type == "clothing":
        product_data['size'] = input("Enter size: ")
        product_data['color'] = input("Enter color: ")
    else:
        print("Invalid product type")
        return

    product_id = self.order_processor.create_product(admin_id, product_data)
    if product_id != -1:
        print(f"Product created successfully with ID: {product_id}")
    else:
```

```
            print("Failed to create product")

    def create_order(self):
        print("\nCreate New Order")
        user_id = int(input("Enter your user ID: "))

        product_ids = [int(pid.strip()) for pid in input("Enter product IDs to order (comma-separated):
").split(',')]

        if self.order_processor.create_order(user_id, product_ids):
            print("Order created successfully!")
        else:
            print("Failed to create order")

    def cancel_order(self):
        print("\nCancel Order")
        user_id = int(input("Enter your user ID: "))
        order_id = int(input("Enter order ID to cancel: "))

        if self.order_processor.cancel_order(user_id, order_id):
            print("Order cancelled successfully!")
        else:
            print("Failed to cancel order")

    def get_all_products(self):
        print("\nAll Products")
        products = self.order_processor.get_all_products()
        for product in products:
            print(f"ID: {product['productId']}, Name: {product['productName']}, Price: {product['price']}")

    def get_orders_by_user(self):
        print("\nOrders by User")
        user_id = int(input("Enter user ID: "))
        orders = self.order_processor.get_order_by_user(user_id)
        for order in orders:
            print(f"ID: {order['productId']}, Name: {order['productName']}")


if __name__ == "__main__":
    app = MainModule()
    app.run()
```

System Working:

```
Order Management System
1. Create User
2. Create Product (Admin only)
3. Create Order
4. Cancel Order
5. Get All Products
6. Get Orders by User
7. Exit
Enter your choice:
```

Create User:

```
Create New User
Enter username: Musfira
Enter password: donuts
Enter role (Admin/User): admin
User created successfully with ID: 7
```

userId	username	password	role
7	Musfira	donuts	Admin

Create Product: (ELECTRONICS)

```
Create New Product (Admin Only)
Enter your admin user ID: 7
Enter product type (Electronics/Clothing): Electronics
Enter product name: Laptop
Enter description: Fast processing high end laptop
Enter price: 10000
Enter quantity in stock: 5
Enter brand: Asus
Enter warranty period (months): 24
Product created successfully with ID: 3
```

productI	productNam	description	price	quantityInS	type	brand	warrantyPeriod	size	color
3	Laptop	Fast processing...	10000.00	5	Electronics	Asus	24	NULL	NULL

Create Product: (CLOTHING)

```
Create New Product (Admin Only)
Enter your admin user ID: 7
Enter product type (Electronics/Clothing): clothing
Enter product name: Floral Dress
Enter description: A long flow dresss
Enter price: 2000
Enter quantity in stock: 5
Enter size: M
Enter color: Red
Product created successfully with ID: 4
```

product	productNam	description	price	quant	type	bra	warra	size	color
4	Floral Dress	A long flow d...	2000...	5	Clot...	NULL	NULL	M	Red

Create Order:

```
Create New Order
Enter your user ID: 7
Enter product IDs to order: 3
Order created successfully!
```

orderId	userId	orderDate	status
5	7	2025-04-10 09:17:26	Pending

orderDetailId	orderId	productId	quantity
5	5	3	1

Cancel Order:

```
Cancel Order
Enter your user ID: 7
Enter order ID to cancel: 3
Order cancelled successfully!
```

orderId	userId	orderDate	status
2	2	2025-04-10 03:37:45	Pending
5	7	2025-04-10 09:17:26	Pending

Get All Prodcuts:

```
All Products
ID: 1, Name: lap, Price: 222.00
ID: 2, Name: sada, Price: 241.00
ID: 3, Name: Laptop, Price: 10000.00
ID: 4, Name: Floral Dress, Price: 2000.00
```

Get Orders by User:

```
Orders by User
Enter user ID: 7
ID: 3, Name: Laptop
```