CODING CHALLENGE: ORDER MANAGEMENT

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 \geq = 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 \geq = 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
  (a) abstractmethod
  def create product(self, user: User, product: Product) -> bool:
    pass
  @abstractmethod
  def create order(self, user: User, products: List[Product]) -> bool:
  (a) abstractmethod
  def cancel order(self, user id: int, order id: int) -> bool:
  @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
       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:
     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]]:
       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

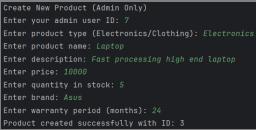
7. Exit
Enter your choice:

Create User:

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

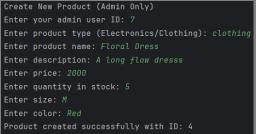
userId	username	password	role
7	Musfira	donuts	Admin

Create Product: (ELECTRONICS)



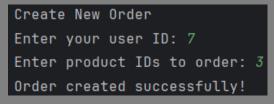
productI	productNami	description	price	quantityInS	type	brand	warrantyPeriod	size	color
3	Laptop	Fast processing	10000.00	5	Electronics	Asus	24	NULL	NULL

Create Product: (CLOTHING)



product	productName	description	price	quant	type	bra warr	size	color
4	Floral Dress	A long flow d	2000	5	Clot	NULL NULL	М	Red

Create Order:



orderId	userI	d orderDa	orderDate		
5	7	2025-04	2025-04-10 09:17:26		
orderDet	tailId	orderId	productId	quantity	
5 5		_	_	_	

Cancel Order:

Cance	l 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