ORDER MANAGEMENT SYSTEM

Project Overview

The Order Management System is a console-based application that helps manage users, products, and customer orders. Admins can create users and products, while customers can place and cancel orders. The system ensures data integrity using a MySQL backend and follows Object-Oriented Programming principles.

Features

- Create user accounts with roles (Admin/User)
- Add new products to the catalog
- Place orders by selecting one or more products
- Cancel placed orders
- View products and orders by user
- Error handling with custom exceptions
- Relational database with foreign key constraints

Technologies Used

Component	Technology
Programming Lang	Python 3
Database	MySQL
Library	mysql-connector-python
Design Pattern	DAO (Data Access Object)
Paradigm	Object-Oriented Programming (OOP)

Task 1: Database Schema Implementation

Tables:

```
-- Users table
CREATE TABLE Users (
    userId INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(50) NOT NULL,
    role VARCHAR(10) NOT NULL CHECK (role IN ('Admin', 'User'))
);

-- Products table (base)
CREATE TABLE Products (
    productId INT PRIMARY KEY AUTO_INCREMENT,
    productName VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL,
    quantityInStock INT NOT NULL,
    type VARCHAR(20) NOT NULL CHECK (type IN ('Electronics', 'Clothing'))
);

-- Electronics table (extends Products)
CREATE TABLE Electronics (
```

```
    productId INT PRIMARY KEY,
    brand VARCHAR(50) NOT NULL,
    warrantyPeriod INT NOT NULL
);

-- Clothing table (extends Products)
CREATE TABLE Clothing (
    productId INT PRIMARY KEY,
    size VARCHAR(10) NOT NULL,
    color VARCHAR(20) NOT NULL
);

-- Orders and OrderDetails tables
CREATE TABLE Orders (
    orderId INT PRIMARY KEY AUTO_INCREMENT,
    userId INT NOT NULL
);

CREATE TABLE OrderDetails (
    orderDetailId INT PRIMARY KEY AUTO_INCREMENT,
    orderId INT NOT NULL,
    productId INT NOT NULL,
    quantity INT NOT NULL
);
```

Task 2: Entity Classes
Location: entities/
Classes:

1. **Product (Base class)**
    - **Attributes: productId, name, description, price, quantity, type**

```
class Product:
    def __init__(self, product_id=0, product_name="", description="", price=0.0, quantity_in_stock=0, 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.type = type

    # Getters and setters
    @property
    def product_id(self):
        return self._product_id

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

```
    @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):
        self._price = value

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

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

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

    @type.setter
    def type(self, value):
        self._type = value

    def _str_(self):
        return f"Product(ID: {self.product_id}, Name: {self.product_name}, Type:
{self.type}, Price: {self.price}, Stock: {self.quantity_in_stock})"
```

2. **Electronics (extends Product)**
    - **Added: brand, warrantyPeriod**

```
from entities.product import Product
```

```
class Electronics(Product):
    def __init__(self, product_id=0, product_name="", description="", price=0.0, quantity_in_stock=0, brand="", warranty_period=0):
        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"
```

3. **Clothing (extends Product)**
   - **Added: size, color**

```
from entities.product import Product

class Clothing(Product):
    def __init__(self, product_id=0, product_name="", description="", price=0.0, quantity_in_stock=0, 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}"
```

4. **User**
   - **Attributes: userId, username, password, role**

```
class User:
    def _init_(self, user_id=0, username="", password="", role=""):
        self.user_id = user_id
        self.username = username
        self.password = password
        self.role = role

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

    @user_id.setter
    def user_id(self, value):
        self._user_id = value

    @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):
        self._role = value

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

Task 3: Repository Interface
File: dao/order_repository.py
Interface Methods:

```
from abc import ABC, abstractmethod
from typing import List
from entities.product import Product
from entities.user import User

class IOrderManagementRepository(ABC):
    @abstractmethod
    def create_order(self, user: User, products: List[Product]):
        pass

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

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

    @abstractmethod
    def create_user(self, user: User):
        pass

    @abstractmethod
    def get_all_products(self) -> List[Product]:
        pass

    @abstractmethod
    def get_order_by_user(self, user: User) -> List[Product]:
        pass
```

Task 4: Exception Handling
Location: exceptions/
Custom Exceptions:

1. UserNotFoundException

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

2. OrderNotFoundException
```
class OrderNotFoundException(Exception):
    def _init_(self, message="Order not found"):
        self.message = message
        super()._init_(self.message)
```

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

Task 5: Database Utilities
Location: utils/
Key Components:

1. DBPropertyUtil: Reads database config
```
import configparser
import os

class DBPropertyUtil:
    @staticmethod
    def get_connection_string(property_file):
        config = configparser.ConfigParser()

        # Get the absolute path to the property file
        current_dir = os.path.dirname(os.path.abspath(_file_))
        property_path = os.path.join(current_dir, "..", property_file)

        config.read(property_path)

        if 'database' in config:
            db_config = config['database']
            return {
                'host': db_config.get('host', 'localhost'),
                'port': db_config.get('port', '3306'),
                'database': db_config.get('database', 'OrderManagementSystem'),
                'user': db_config.get('username', 'root'),
                'password': db_config.get('password', '')
            }
        else:
            raise Exception("Database configuration not found in property file")
```

2. DBConnUtil: Manages connections
```
import mysql.connector
from mysql.connector import Error
from utils.db_property_util import DBPropertyUtil

class DBConnUtil:
```

```
    @staticmethod
    def get_connection(property_file='db.properties'):
        try:
            # Get connection properties
            connection_properties = DBPropertyUtil.get_connection_string(property_file)

            # Establish connection
            connection = mysql.connector.connect(
                host=connection_properties['host'],
                port=connection_properties['port'],
                database=connection_properties['database'],
                user=connection_properties['user'],
                password=connection_properties['password']
            )

            if connection.is_connected():
                print("Successfully connected to the database")
                return connection

        except Error as e:
            print(f"Error while connecting to MySQL: {e}")
            raise

    @staticmethod
    def close_connection(connection):
        if connection and connection.is_connected():
            connection.close()
            print("MySQL connection is closed")
```

Task 6: Main Application

File: main.py

Features:

- Menu-driven interface
- Transaction management
- User input validation

```
from typing import List
from entities.product import Product
from entities.electronics import Electronics
from entities.clothing import Clothing
from entities.user import User
from exceptions.user_not_found import UserNotFoundException
from exceptions.order_not_found import OrderNotFoundException
from dao.order_processor import OrderProcessor

def display_menu():
    print("\n===== Order 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")
    return input("Enter your choice: ")

def create_user(order_processor: OrderProcessor):
    print("\n--- Create New User ---")
    username = input("Enter username: ")
    password = input("Enter password: ")
    role = input("Enter role (Admin/User): ")

    user = User(0, username, password, role)
    order_processor.create_user(user)

def create_product(order_processor: OrderProcessor):
    print("\n--- Create New Product ---")

    user_id = int(input("Enter admin user ID: "))
    product_type = input("Enter product type (Electronics/Clothing): ").capitalize()

    name = input("Enter product name: ")
    description = input("Enter description: ")
    price = float(input("Enter price: "))
    quantity = int(input("Enter quantity in stock: "))

    user = User(user_id)

    if product_type == "Electronics":
        brand = input("Enter brand: ")
        warranty = int(input("Enter warranty period (months): "))
        product = Electronics(0, name, description, price, quantity, brand, warranty)
    else:
        size = input("Enter size: ")
        color = input("Enter color: ")
        product = Clothing(0, name, description, price, quantity, size, color)

    try:
        order_processor.create_product(user, product)
    except UserNotFoundException as e:
        print(f"Error: {e}")

def create_order(order_processor: OrderProcessor):
    print("\n--- Create New Order ---")

    user_id = int(input("Enter user ID: "))
    products = []
```

```
    while True:
        product_id = input("Enter product ID to add to order (or 'done' to finish): ")
        if product_id.lower() == 'done':
            break

        try:
            product_id = int(product_id)
            product = Product(product_id)
            products.append(product)
        except ValueError:
            print("Please enter a valid product ID or 'done'")

    if products:
        user = User(user_id)
        try:
            order_processor.create_order(user, products)
        except UserNotFoundException as e:
            print(f"Error: {e}")

def cancel_order(order_processor: OrderProcessor):
    print("\n--- Cancel Order ---")

    user_id = int(input("Enter user ID: "))
    order_id = int(input("Enter order ID to cancel: "))

    try:
        order_processor.cancel_order(user_id, order_id)
    except (UserNotFoundException, OrderNotFoundException) as e:
        print(f"Error: {e}")

def get_all_products(order_processor: OrderProcessor):
    print("\n--- All Products ---")
    products = order_processor.get_all_products()

    if not products:
        print("No products available.")
    else:
        for product in products:
            print(product)

def get_orders_by_user(order_processor: OrderProcessor):
    print("\n--- Orders by User ---")

    user_id = int(input("Enter user ID: "))
    user = User(user_id)

    try:
        products = order_processor.get_order_by_user(user)
```

```
        if not products:
            print("No orders found for this user.")
        else:
            print(f"Orders for user ID {user_id}:")
            for product in products:
                print(product)
    except UserNotFoundException as e:
        print(f"Error: {e}")

def main():
    order_processor = OrderProcessor()

    while True:
        choice = display_menu()

        try:
            if choice == '1':
                create_user(order_processor)
            elif choice == '2':
                create_product(order_processor)
            elif choice == '3':
                create_order(order_processor)
            elif choice == '4':
                cancel_order(order_processor)
            elif choice == '5':
                get_all_products(order_processor)
            elif choice == '6':
                get_orders_by_user(order_processor)
            elif choice == '7':
                print("Exiting the system. Goodbye!")
                break
            else:
                print("Invalid choice. Please try again.")
        except Exception as e:
            print(f"An error occurred: {e}")

if _name_ == "_main_":
    main()
```

Sample Outputs

1. Create User

```
===== 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: 1

--- Create New User ---
Enter username: test_user
Enter password: password123
Enter role (Admin/User): User
User created successfully!
```

2. Create Product (Admin)

```
Enter your choice: 2

--- Create New Product ---
Enter admin user ID: 1
Enter product type (Electronics/Clothing): Electronics
Enter product name: Smart Watch
Enter description: Fitness tracker
Enter price: 199.99
Enter quantity in stock: 50
Enter brand: Apple
Enter warranty period (months): 12
Product created successfully!
```

3. Create Order

```
Enter your choice: 3

--- Create New Order ---
Enter user ID: 2
Enter product ID to add to order (or 'done' to finish): 1
Enter product ID to add to order (or 'done' to finish): 3
Enter product ID to add to order (or 'done' to finish): done
Order created successfully!
```

4. View All Products

```
Enter your choice: 5

--- All Products ---
1: Laptop (Electronics) - $999.99 - Stock: 9
   Brand: Dell, Warranty: 24 months
2: Smartphone (Electronics) - $699.99 - Stock: 15
   Brand: Samsung, Warranty: 12 months
3: T-Shirt (Clothing) - $19.99 - Stock: 48
   Size: M, Color: Black
4: Jeans (Clothing) - $49.99 - Stock: 29
   Size: 32, Color: Blue
5: Smart Watch (Electronics) - $199.99 - Stock: 50
   Brand: Apple, Warranty: 12 months
```

5. View User Orders

```
Enter your choice: 6

--- Orders by User ---
Enter user ID: 2
Orders for user ID 2:
1: Laptop (Electronics) - $999.99
3: T-Shirt (Clothing) - $19.99
```

6. Cancel Order

```
Enter your choice: 4

--- Cancel Order ---
Enter user ID: 2
Enter order ID to cancel: 3
Order cancelled successfully!
```