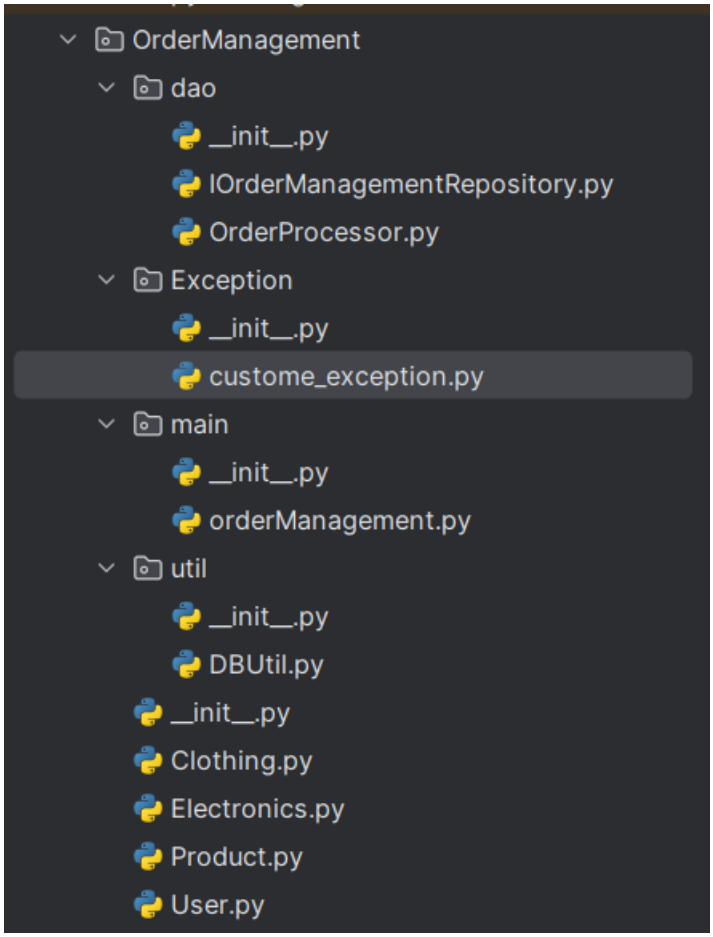
Coding Challenge - Order Management System

Submitted by Sankar Roy

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.

© Hexaware Technologies Limited. All rights



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]

© Hexaware Technologies Limited. All rights

```python
class Product:

    def __init__(self, productId, productName, description, price, quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.setType(type)

    @property
    def getProductId(self):
        return self.productId

    @property
    def getProductName(self):
        return self.productName

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

    @property
    def getPrice(self):
        return self.price
```

```python
    @property                                                    ⚠ 27
    def getQuantityInStock(self):
        return self.quantityInStock

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

    def setProductId(self, productId):
        self.productId = productId

    def setProductName(self, productName):
        self.productName = productName

    def setDescription(self, description):
        self.description = description

    def setPrice(self, price):
        self.price = price

    def setQuantityInStock(self, quantityInStock):
        self.quantityInStock = quantityInStock

    1 usage
    def setType(self, type):
        if type not in ['Electronics', 'Clothing']:
            raise Exception("You have entered wrong type: please choose between Electronics and Clothing")
        self.type = type
```

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

© Hexaware Technologies Limited. All rights

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

```python
from OrderManagement.Product import Product

1 usage
class Electronics(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, brand, warrantyPeriod):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.brand = brand
        self.warrantyPeriod = warrantyPeriod

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

    def setBrand(self, brand):
        self.brand = brand

    1 usage
    @property
    def getWarrantyPeriod(self):
        return self.warrantyPeriod

    def setWarrantyPeriod(self, warrantyPeriod):
        self.warrantyPeriod = warrantyPeriod
```

© Hexaware Technologies Limited. All rights

- **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)

```python
from OrderManagement.Product import Product

1 usage
class Clothing(Product):
    def __init__(self, productId, productName, description, price, quantityInStock, type, size, color):
        super().__init__(productId, productName, description, price, quantityInStock, type)
        self.size = size
        self.color = color

    1 usage
    @property
    def getSize(self):
        return self.size

    @getSize.setter
    def setSize(self, size):
        self.size = size

    2 usages
    @property
    def getColor(self):
        return self.color

    @getColor.setter
    def setColor(self, color):
        self.color = color
```

5. Create a **User** class with attributes:
    - **userId** (int)
    - **username** (String)
    - **password** (String)
    - **role** (String) // "Admin" or "User"

© Hexaware Technologies Limited. All rights

```python
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        self.setRole(role)

    @property
    def getUserId(self):
        return self.userId

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

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

    1 usage
    @property
    def getRole(self):
        return self.role


    def setUserId(self, userId):
        self.userId = userId
```

```python
  def setUsername(self, username):
      self.username = username

  def setPassword(self, password):
      self.password = password

  2 usages
  def setRole(self, role):
      if role not in ['Admin', 'User']:
          raise Exception("The role should be Admin or User")
      self.role = role
```

6. Define an interface/abstract class named **IOrderManagementRepository** with methods for:

© Hexaware Technologies Limited. All rights

- **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.

```python
from abc import ABC, abstractmethod


class IOrderManagementRepository(ABC):
    @abstractmethod
    def createOrder(self):
        pass

    1 usage (1 dynamic)
    @abstractmethod
    def cancelOrder(self, userId, orderId):
        pass

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

    @abstractmethod
    def createUser(self, user):
        pass

    @abstractmethod
    def getAllProducts(self):
        pass

    @abstractmethod
    def getOrderByUser(self, userID):
        pass
```

7. Implement the **IOrderManagementRepository** interface/abstractclass in a class called **OrderProcessor**. This class will be responsible for managing orders.

© Hexaware Technologies Limited. All rights

```
def createOrder(self):
    global conn
    try:
        conn = self.dbutil.getDBConnection()
        cursor = conn.cursor()

        orderId = int(input("Enter the order ID: "))
        productId = int(input("Enter the product ID: "))
        userId = int(input("Enter the user ID: "))

        cursor.execute("INSERT INTO orders (orderId, productId, userId) VALUES (%s, %s, %s) "
                       "RETURNING orderId", (orderId, productId, userId))
        order_id = cursor.fetchone()[0]
        conn.commit()

        cursor.close()
        conn.close()

        return order_id
    except Exception as e:
        print("Error creating order:", e)
        conn.rollback()
        return None
```

```
1 usage (1 dynamic)
def cancelOrder(self, orderId, userId):
    try:
        conn = self.dbutil.getDBConnection()
        cursor = conn.cursor()

        cursor.execute("SELECT orderId FROM orders WHERE orderId = %s AND userId = %s", (orderId, userId))
        if not cursor.fetchone():
            raise OrderNotFound(f"Order with ID {orderId} not found for user with ID {userId}")

        cursor.execute("DELETE FROM orders WHERE orderId = %s AND userId = %s", (orderId, userId))
        conn.commit()

        cursor.close()
        conn.close()

        print("Order successfully cancelled.")
    except OrderNotFound as e:
        print(e)
    except Exception as e:
        print("Error cancelling order:", e)
        conn.rollback()
```

© Hexaware Technologies Limited. All rights

```
1 usage                                                                    ⚠2 ⚠23
def createProduct(self, user, product):
    global conn
    try:
        conn = self.dbutil.getDBConnection()
        cursor = conn.cursor()


        cursor.execute("SELECT role FROM users WHERE userId = %s", (user.userId,))
        user_role = cursor.fetchone()
        if not user_role or user_role[0] != "Admin":
            raise UserNotFound("User is not an admin.")

        cursor.execute(
            "INSERT INTO products (productId, productName, description,"
            " price, quantityInStock, type) VALUES (%s, %s, %s, %s, %s, %s)",
            (product.productId, product.productName, product.description, product.price, product.quantityInStock,
             product.type))
        conn.commit()

        cursor.close()
        conn.close()

        print("Product created successfully.")
    except UserNotFound as e:
        print(e)
    except Exception as e:
        print("Error creating product:", e)
        conn.rollback()
```

```
1 usage
def getAllProducts(self):
    try:
        conn = self.dbutil.getDBConnection()
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM products")
        products = cursor.fetchall()

        cursor.close()
        conn.close()

        print("All products retrieved successfully.")
        return products
    except Exception as e:
        print("Error retrieving products:", e)
        return None
```

© Hexaware Technologies Limited. All rights

```
1 usage
def getOrderByUser(self, user):
    try:
        conn = self.dbutil.getDBConnection()
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM orders WHERE userId = %s", (user.userId,))
        orders = cursor.fetchall()

        cursor.close()
        conn.close()

        print(f"All orders for user {user.username} retrieved successfully.")
        return orders
    except Exception as e:
        print("Error retrieving orders:", e)
        return None
```

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

```
import mysql.connector


class DBUtil:
    def __init__(self):
        self.con = mysql.connector.connect(
            host="localhost",
            port="3306",
            user="root",
            password="root",
            database="ordermanagement"
        )

    6 usages (6 dynamic)
    def getDBConnection(self):
        return self.con.cursor()
```

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",

© Hexaware Technologies Limited. All rights

"cancelOrder", "getAllProducts", "getOrderbyUser", "exit".

```python
from OrderManagement.Product import Product
from OrderManagement.dao.OrderProcessor import OrderProcessor
from OrderManagement.User import User
1 usage
class MainModule(OrderProcessor):


    1 usage
    def display_menu(self):
        print("Menu:")
        print("1. Create User")
        print("2. Create Product")
        print("3. Create Order")
        print("4. Cancel Order")
        print("5. Get All Products")
        print("6. Get Order by User")
        print("7. Exit")
```

© Hexaware Technologies Limited. All rights

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

        if choice == "1":
            self.OrderProcessor.create_user(User)
        elif choice =="2":
            self.createUser(User)
        elif choice == "3":
            userID = int(input("enter the userID : "))
            orderID = int(input("enter the order id : "))
            self.OrderProcessor.cancelOrder(orderID,userID)
        elif choice =="4":
            self.createProduct(User, Product)
        elif choice =="5":
            self.getAllProducts()
        elif choice =="6":
            self.getOrderByUser(User)
        elif choice == "7":
            print("exiting from menu")
            break
        else :
            print("enter correct choice")
```

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

© Hexaware Technologies Limited. All rights