ORDER MANAGEMENT

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]

```
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.type = type  # Either "Electronics" or "Clothing"

    def __str__(self):
        return f"Product ID: {self.productId}, Name: {self.productName}, Description: {self.description}, Price: {self.price}, Quantity in Stock: {self.quantityInStock}, Type: {self.type}"

    def get_product_id(self):
        return self.productId

    def set_product_id(self, productId):
        self.productId = productId
```

```
def get_product_name(self):
    return self.productName

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

def get_description(self):
    return self.description

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

def get_price(self):
    return self.price

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

def get_quantity_in_stock(self):
    return self.quantityInStock

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

def get_type(self):
    return self.type

def set_type(self, type):
    self.type = type
```

```
product1 = Product(1, "Laptop", "High-performance laptop", 999.99, 10, "Electronics")
print(product1)
```

```
Product ID: 1, Name: Laptop, Description: Mac, Price: 999.99, Quantity in Stock: 10, Type: Electronics

Process finished with exit code 0
```

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

```
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.type = type


    def get_product_id(self):
        return self.productId

    def get_product_name(self):
        return self.productName

    def get_description(self):
        return self.description

    def get_price(self):
```

```
    return self.price

def get_quantity_in_stock(self):
    return self.quantityInStock

def get_type(self):
    return self.type


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

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

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

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

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

def set_type(self, type):
    self.type = type


@classmethod
def from_default(cls):
```

```
        return cls(None, None, None, None, None, None)



product1 = Product(1, "Laptop", "High-performance laptop", 999.99, 10, "Electronics")
print(product1.get_product_name())

product1.set_price(899.99)
print(product1.get_price())



default_product = Product.from_default()
print(default_product.get_product_name())
```

```
Laptop
899.99
None

Process finished with exit code 0
```

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

• brand (String)

• warrantyPeriod (int)

```
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


    def get_brand(self):
        return self.brand

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

    def get_warranty_period(self):
        return self.warrantyPeriod

    def set_warranty_period(self, warrantyPeriod):
        self.warrantyPeriod = warrantyPeriod



electronics_product = Electronics(2, "Smartphone", "High-end smartphone", 899.99, 20, "Electronics", "Samsung", 1)
print(electronics_product.get_brand())

electronics_product.set_warranty_period(2)
print(electronics_product.get_warranty_period())
```

```
Laptop
899.99
None
Samsung
2
```

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

• size (String)

• color (String)

```
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



    def get_size(self):
        return self.size

    def set_size(self, size):
        self.size = size

    def get_color(self):
        return self.color

    def set_color(self, color):
        self.color = color



clothing_product = Clothing(3, "T-shirt", "Casual t-shirt", 19.99, 50, "Clothing", "M", "Blue")
```

```
print(clothing_product.get_size())

clothing_product.set_color("Red")
print(clothing_product.get_color())
```

```
M
Red

Process finished with exit code 0
```

5. Create a User class with attributes:

• userId (int)

• username (String)

• password (String)

• role (String) // "Admin" or "User"

```
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        self.role = role  # Either "Admin" or "User"



    def get_user_id(self):
        return self.userId

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

```
    def get_username(self):
        return self.username

    def set_username(self, username):
        self.username = username

    def get_password(self):
        return self.password

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

    def get_role(self):
        return self.role

    def set_role(self, role):
        self.role = role


user1 = User(1, "admin", "admin@123", "Admin")
print(user1.get_username())

user2 = User(2, "user1", "user123", "User")
print(user2.get_role())
```

```
admin
User

Process finished with exit code 0
```

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

• 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.

```
from abc import ABC, abstractmethod

class IOrderManagementRepository(ABC):
    @abstractmethod
    def create_order(self, user, products):
        pass

    @abstractmethod
    def cancel_order(self, userId, orderId):
        pass

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

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

```
    @abstractmethod
    def get_all_products(self):
        pass

    @abstractmethod
    def get_order_by_user(self, user):
        pass



class OrderManagementRepositoryImpl(IOrderManagementRepository):
    def create_order(self, user, products):
        pass

    def cancel_order(self, userId, orderId):
        pass

    def create_product(self, user, product):
        pass

    def create_user(self, user):
        pass

    def get_all_products(self):
        pass

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

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

```
class OrderProcessor(IOrderManagementRepository):
    def __init__(self):
        self.users_database = {}
        self.orders_database = {}
        self.products_database = {}

    def createOrder(self, user, products):
        user_id = user.get_user_id()
        if user_id not in self.users_database:
            self.createUser(user)
        order_id = len(self.orders_database) + 1
        self.orders_database[order_id] = {'user_id': user_id, 'products': products}
        return order_id

    def cancelOrder(self, userId, orderId):
        if orderId not in self.orders_database or self.orders_database[orderId]['user_id'] != userId:
            raise OrderNotFound("Order not found or does not belong to the specified user.")
        del self.orders_database[orderId]

    def createProduct(self, user, product):
        if user.get_role() != "Admin":
            raise PermissionError("Only admin users can create products.")
        product_id = len(self.products_database) + 1
        self.products_database[product_id] = product

    def createUser(self, user):
        user_id = user.get_user_id()
```

```
        if user_id not in self.users_database:
            self.users_database[user_id] = user

    def getAllProducts(self):
        return list(self.products_database.values())

    def getOrderByUser(self, user):
        user_id = user.get_user_id()
        user_orders = []
        for order_id, order_info in self.orders_database.items():
            if order_info['user_id'] == user_id:
                user_orders.append(order_info)
        return user_orders
```

```
Product ID: 1, Name: Laptop, Description: High-performance laptop, Price: 999.99, Quantity in Stock: 10, Type: Electronics
Laptop
899.99
```

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
from util.PropertyUtil import PropertyUtil


class DBConnection:
    connection = None
```

```
def getConnection():
    if DBConnection.connection is None:
        connection_string = PropertyUtil.getPropertyString()
        try:
            DBConnection.connection = mysql.connector.connect(**connection_string)
            print("Connection successful")
        except mysql.connector.Error as error:
            print("Error while connecting to MySQL", error)
    return DBConnection.connection
```

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

```
class OrderProcessor(IOrderManagementRepository):
    def __init__(self):
        self.users_database = {}
        self.orders_database = {}
        self.products_database = {}

    def createOrder(self, user, products):
        user_id = user.get_user_id()
        if user_id not in self.users_database:
            self.createUser(user)
        order_id = len(self.orders_database) + 1
        self.orders_database[order_id] = {'user_id': user_id, 'products': products}
        return order_id
```

```
def cancelOrder(self, userId, orderId):
    if orderId not in self.orders_database or self.orders_database[orderId]['user_id'] != userId:
        raise OrderNotFound("Order not found or does not belong to the specified user.")
    del self.orders_database[orderId]

def createProduct(self, user, product):
    if user.get_role() != "Admin":
        raise PermissionError("Only admin users can create products.")
    product_id = len(self.products_database) + 1
    self.products_database[product_id] = product

def createUser(self, user):
    user_id = user.get_user_id()
    if user_id not in self.users_database:
        self.users_database[user_id] = user

def getAllProducts(self):
    return list(self.products_database.values())

def getOrderByUser(self, user):
    user_id = user.get_user_id()
    user_orders = []
    for order_id, order_info in self.orders_database.items():
        if order_info['user_id'] == user_id:
            user_orders.append(order_info)
    return user_orders
```

```
Process finished with exit code 0
```