Coding Challenge – 6 : Order Management System (Meghanath P.)

Python Code:

```
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="rogerdeakins",
            database="oms"
        )
        return connection


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


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


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


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


class UserNotFound(Exception):
    pass
```

```
class OrderNotFound(Exception):
    pass


class IOrderManagementRepository:
    @staticmethod
    def createOrder(user, products):
        try:
            # Checking here if the user exists in the database or not
            connection = DBUtil.getDBConn()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM User WHERE userId = %s",
(user.userId,))
            existing_user = cursor.fetchone()

            if existing_user is None:
                # and if the User is not found, creating  the user
                cursor.execute("INSERT INTO User (userId, username,
password, role) VALUES (%s, %s, %s, %s)",
                               (user.userId, user.username, user.password,
user.role))
                connection.commit()

            # Creating the order
            for product in products:
                cursor.execute("INSERT INTO ProductOrder (userId,
productId) VALUES (%s, %s)",
                               (user.userId, product.productId))
                connection.commit()

            connection.close()
        except Exception as e:
            print(f"Error creating order: {e}")
            raise

    @staticmethod
    def cancelOrder(userId, orderId):
        try:
            # Checking here if the user exists in the database or not
            connection = DBUtil.getDBConn()
            cursor = connection.cursor()

            cursor.execute("SELECT * FROM User WHERE userId = %s",
(userId,))
            existing_user = cursor.fetchone()

            if existing_user is None:
                raise UserNotFound("User not found")

            cursor.execute("SELECT * FROM ProductOrder WHERE userId = %s
AND orderId = %s", (userId, orderId))
            existing_order = cursor.fetchone()

            if existing_order is None:
                raise OrderNotFound("Order not found")

            # Cancelling the order
            cursor.execute("DELETE FROM ProductOrder WHERE userId = %s AND
orderId = %s", (userId, orderId))
```

```
            connection.commit()

            connection.close()
        except UserNotFound as e:
            print(f"UserNotFound: {e}")
            raise
        except OrderNotFound as e:
            print(f"OrderNotFound: {e}")
            raise
        except Exception as e:
            print(f"Error cancelling order: {e}")
            raise

    @staticmethod
    def createProduct(admin_user, product):
        try:
            # Checking here if the user exists in the database or not
            connection = DBUtil.getDBConn()
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM User WHERE userId = %s AND role =
'Admin'", (admin_user.userId,))
            existing_admin = cursor.fetchone()

            if existing_admin is None:
                raise UserNotFound("Admin user not found")

            # Creating the product
            cursor.execute("INSERT INTO Product (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))
            connection.commit()

            # Checking if it is a electronics product or a clothing product
            if product.type == 'Electronics':
                cursor.execute("INSERT INTO Electronics (productId, brand,
warrantyPeriod) "
                               "VALUES (%s, %s, %s)",
                               (product.productId, product.brand,
product.warrantyPeriod))
            elif product.type == 'Clothing':
                cursor.execute("INSERT INTO Clothing (productId, size,
color) "
                               "VALUES (%s, %s, %s)",
                               (product.productId, product.size,
product.color))

            connection.commit()
            connection.close()
        except UserNotFound as e:
            print(f"UserNotFound: {e}")
            raise
        except Exception as e:
            print(f"Error creating product: {e}")
            raise

    @staticmethod
    def createUser(user):
        try:
```

```
            # Creating the user
            conn = DBUtil.getDBConn()
            cursor = conn.cursor()
            cursor.execute("INSERT INTO User (userId, username, password,
role) VALUES (%s, %s, %s, %s)",
                           (user.userId, user.username, user.password,
user.role))
            conn.commit()
            conn.close()
        except Exception as e:
            print(f"Error creating user: {e}")
            raise

    @staticmethod
    def getAllProducts():
        try:

            # getting all products from the database
            conn = DBUtil.getDBConn()
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT * FROM Product")
            products = cursor.fetchall()
            conn.close()
            return products
        except Exception as e:
            print(f"Error getting all products: {e}")
            raise

    @staticmethod
    def getOrderByUser(user):
        try:
            # getting all products ordered by a specific user
            conn = DBUtil.getDBConn()
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT p.* FROM Product p "
                           "JOIN ProductOrder po ON p.productId =
po.productId "
                           "WHERE po.userId = %s",
                           (user.userId,))
            ordered_products = cursor.fetchall()
            conn.close()
            return ordered_products
        except Exception as e:
            print(f"Error getting ordered products: {e}")
            raise


class OrderProcessor(IOrderManagementRepository):
    pass
```