CODING CHALLENGE

Submitted by: Fahimunnisha A

Topic: Order Management System

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:

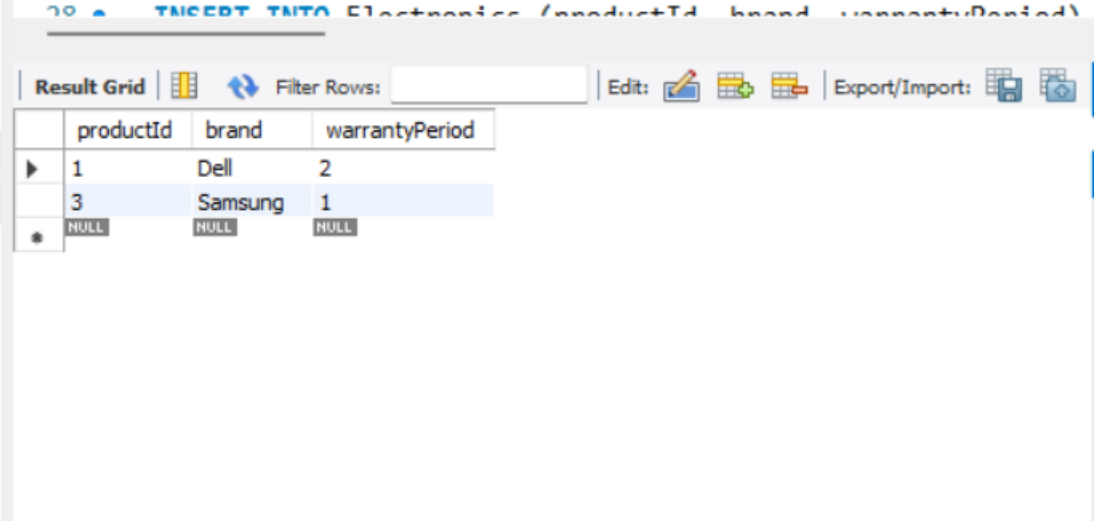
- productId (int)
- productName (String)
- description (String)
- price (double)
- quantityInStock (int)
- type (String) [Electronics/Clothing]

```sql
CREATE TABLE Product (
    productId INT PRIMARY KEY,
    productName VARCHAR(255),
    description TEXT,
    price DOUBLE,
    quantityInStock INT,
    type VARCHAR(50) CHECK (type IN ('Electronics', 'Clothing'))
)
```

Result Grid | Filter Rows: | Edit: | Export/Import:

productId	productName	description	price	quantityInStock	type
1	Laptop	A high-performance laptop	1200.5	10	Electronics
2	T-Shirt	Cotton t-shirt	25.99	50	Clothing
3	Smartphone	Latest model smartphone	899.99	15	Electronics
4	Jeans	Denim jeans	40	30	Clothing
NULL	NULL	NULL	NULL	NULL	NULL

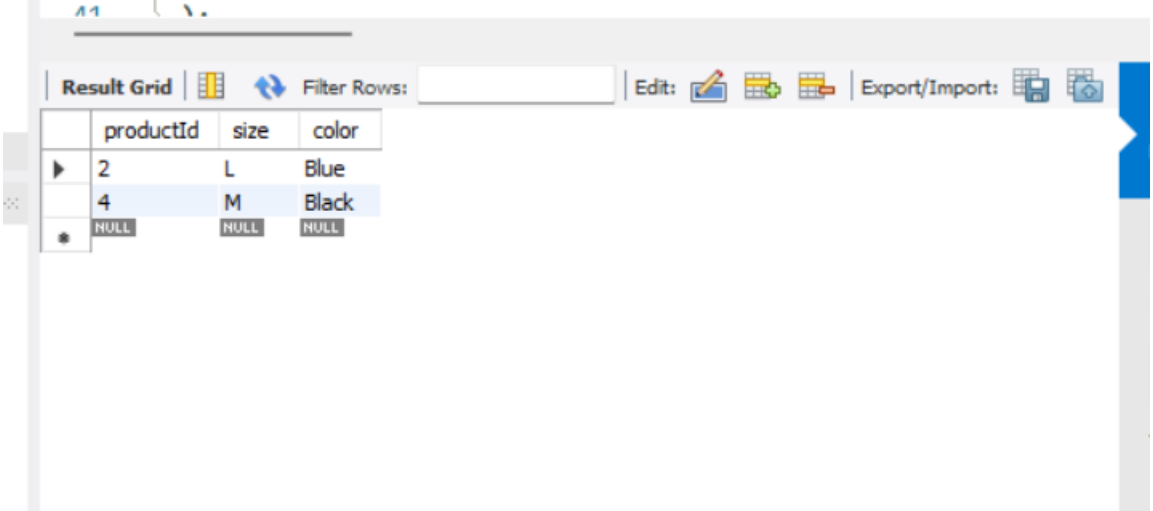
```sql
CREATE TABLE Electronics (
    productId INT PRIMARY KEY,
    brand VARCHAR(255),
    warrantyPeriod INT,
    FOREIGN KEY (productId) REFERENCES Product(productId)
);
```



Result Grid | Filter Rows: | Edit: | Export/Import:

productId	brand	warrantyPeriod
1	Dell	2
3	Samsung	1
NULL	NULL	NULL

```sql
CREATE TABLE Clothing (
    productId INT PRIMARY KEY,
    size VARCHAR(50),
    color VARCHAR(50),
    FOREIGN KEY (productId) REFERENCES Product(productId)
);
```



Result Grid | Filter Rows: | Edit: | Export/Import:

productId	size	color
2	L	Blue
4	M	Black
NULL	NULL	NULL

```sql
CREATE TABLE User (
    userId INT PRIMARY KEY,
    username VARCHAR(255),
    password VARCHAR(255),
    role VARCHAR(50) CHECK (role IN ('Admin', 'User'))
);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

userId	username	password	role
1	admin_user	admin123	Admin
2	john_doe	password123	User
3	jane_smith	password456	User
NULL	NULL	NULL	NULL

```sql
CREATE TABLE Orders (
    orderId INT PRIMARY KEY,
    userId INT,
    productId INT,
    orderDate DATE,
    FOREIGN KEY (userId) REFERENCES User(userId),
```

Result Grid | Filter Rows: | Edit: | Export/Import:

orderId	userId	productId	orderDate
1	2	1	2025-04-17
2	3	2	2025-04-16
3	2	3	2025-04-15
NULL	NULL	NULL	NULL

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

```java
package com.product;

public class Product {
    private int productId;
    private String productName;
    private String description;
    private double price;
    private int quantityInStock;
    private String type; // Electronics/Clothing

    public Product() {}

    public Product(int productId, String productName, String description, double price, int quantityInStock, String type) {
        this.productId = productId;
        this.productName = productName;
        this.description = description;
        this.price = price;
        this.quantityInStock = quantityInStock;
        this.type = type;
    }

    // Getters and Setters for all attributes
    public int getProductId() { return productId; }
    public void setProductId(int productId) { this.productId = productId; }

    public String getProductName() { return productName; }
    public void setProductName(String productName) { this.productName = productName; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }

    public int getQuantityInStock() { return quantityInStock; }
    public void setQuantityInStock(int quantityInStock) { this.quantityInStock = quantityInStock; }

    public String getType() { return type; }
    public void setType(String type) { this.type = type; }
}
```

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

- **brand** (String)
- **warrantyPeriod** (int)

```java
package com.product;

public class Electronics extends Product {
    private String brand;
    private int warrantyPeriod; // in months or years

    // Default constructor
    public Electronics() {
        super();
    }

    // Parameterized constructor
    public Electronics(int productId, String productName, String description, double price, int quantityInStock, String type,
                       String brand, int warrantyPeriod) {
        super(productId, productName, description, price, quantityInStock, type);
        this.brand = brand;
        this.warrantyPeriod = warrantyPeriod;
    }

    // Getters and Setters
    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public int getWarrantyPeriod() {
        return warrantyPeriod;
    }

    public void setWarrantyPeriod(int warrantyPeriod) {
        this.warrantyPeriod = warrantyPeriod;
    }
}
```

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

- **size** (String)
- **color** (String)

```java
package com.product;

public class Clothing extends Product {
    private String size;
    private String color;

    // Default constructor
    public Clothing() {
        super();
    }

    // Parameterized constructor
    public Clothing(int productId, String productName, String description, double price, int quantityInStock, String type,
                    String size, String color) {
        super(productId, productName, description, price, quantityInStock, type);
        this.size = size;
        this.color = color;
    }

    // Getters and Setters
    public String getSize() {
        return size;
    }

    public void setSize(String size) {
        this.size = size;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

5. Create a **User** class with attributes:

- **userId** (int)
- **username** (String)
- **password** (String)
- **role** (String) // "Admin" or "User"

```java
package com.user;
public class User {
    private int userId;
    private String username;
    private String password;
    private String role;
    public User() {}
    public User(int userId, String username, String password, String role) {
        this.userId = userId;
        this.username = username;
        this.password = password;
        this.role = role;
    }

    // Getters and Setters
    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }
```

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.

```java
package com.repository;

import com.user.User;
import com.product.Product;
import java.util.List;

public interface IOrderManagementRepository {

    void createOrder(User user, List<Product> products) throws Exception;

    void cancelOrder(int userId, int orderId) throws Exception;

    void createProduct(User user, Product product) throws Exception;

    void createUser(User user) throws Exception;

    List<Product> getAllProducts() throws Exception;

    List<Product> getOrderByUser(User user) throws Exception;
}
```

```java
package com.exception;


public class AdminNotFoundException extends Exception {
    public AdminNotFoundException(String message) {
        super(message);
    }
}
```

```java
package com.exception;


public class OrderManagementException extends Exception {
 public OrderManagementException(String message) {
     super(message);
 }
}
```

```java
package com.exception;


public class ProductNotFoundException extends Exception {
    public ProductNotFoundException(String message) {
        super(message);
    }
}
```

```java
package com.exception;



public class UserAlreadyExistsException extends Exception {
    public UserAlreadyExistsException(String message) {
        super(message);
    }
}
```

```java
package com.exception;


public class UserNotFoundException extends Exception {
    public UserNotFoundException(String message) {
        super(message);
    }
}
```

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

```java
package com.service;

import com.user.User;
import com.product.Product;
import com.exception.UserNotFoundException;
import com.exception.OrderManagementException;
import com.exception.ProductNotFoundException;
import com.exception.AdminNotFoundException;
import com.exception.UserAlreadyExistsException;
import com.repository.IOrderManagementRepository;
import java.util.List;
import java.util.Optional;
import com.user.Order;
import com.user.User;

public class OrderProcessor implements IOrderManagementRepository {

    // Simulating a database with collections (in real-life, this would be a DB call)
    private List<User> users;
    private List<Product> products;
    private List<Order> orders;

    public OrderProcessor(List<User> users, List<Product> products, List<Order> orders) {
        this.users = users;
        this.products = products;
        this.orders = orders;
    }

    // Implementing the createOrder method
    @Override
    public void createOrder(User user, List<Product> products) throws UserNotFoundException, ProductNotFoundException {
        // Check if user exists in the database
        Optional<User> existingUser = users.stream()
            .filter(u -> u.getId() == user.getId()) // Assuming user ID is unique
            .findFirst();

        if (!existingUser.isPresent()) {
```

```
            createUser(user);
        }

        // Check if the products exist in the database
        for (Product product : products) {
            Optional<Product> existingProduct = products.stream()
                .filter(p -> p.getId() == product.getId()) // Assuming product ID is unique
                .findFirst();

            if (!existingProduct.isPresent()) {
                throw new ProductNotFoundException("Product with ID " + product.getId() + " not found.");
            }
        }

        // Create the order
        Order order = new Order(user, products);
        orders.add(order);
        System.out.println("Order created successfully for user " + user.getName());
    }

    // Implementing cancelOrder
    @Override
    public void cancelOrder(int userId, int orderId) throws UserNotFoundException, OrderManagementException {
        // Check if user exists
        Optional<User> user = users.stream()
            .filter(u -> u.getId() == userId)
            .findFirst();

        if (!user.isPresent()) {
            throw new UserNotFoundException("User with ID " + userId + " not found.");
        }

        // Check if the order exists
        Optional<Order> order = orders.stream()
            .filter(o -> o.getId() == orderId)
            .findFirst();
```

```
    @Override
    public void createProduct(User user, Product product) throws AdminNotFoundException {
        // Check if the user is an admin
        if (!user.isAdmin()) {
            throw new AdminNotFoundException("User is not an admin.");
        }

        // Add product to the database
        products.add(product);
        System.out.println("Product created successfully.");
    }

    // Implementing createUser method to add new users
    @Override
    public void createUser(User user) throws UserAlreadyExistsException {
        // Check if the user already exists in the database
        Optional<User> existingUser = users.stream()
            .filter(u -> u.getId() == user.getId())
            .findFirst();

        if (existingUser.isPresent()) {
            throw new UserAlreadyExistsException("User with ID " + user.getId() + " already exists.");
        }

        // Add new user to the database
        users.add(user);
        System.out.println("User created successfully.");
    }

    // Implementing getAllProducts method to retrieve all products
    @Override
    public List<Product> getAllProducts() {
        return products;
    }

    // Implementing getOrderByUser method to retrieve all orders by a user
    @Override
    public List<Order> getOrderByUser(User user) throws UserNotFoundException {
```

```java
@Override
public List<Order> getOrderByUser(User user) throws UserNotFoundException {
    // Check if the user exists
    Optional<User> existingUser = users.stream()
        .filter(u -> u.getId() == user.getId())
        .findFirst();

    if (!existingUser.isPresent()) {
        throw new UserNotFoundException("User with ID " + user.getId() + " not found.");
    }

    // Get orders by user
    List<Order> userOrders = orders.stream()
        .filter(order -> order.getUser().getId() == user.getId())
        .toList();

    return userOrders;
}
}
```

8. Create **DBUtil** class and add the following method.

- **static getDBConn():Connection** Establish a connection to the database and return database Connection

```java
package com.util;

import java.sql.Connection;

public class DBUtil {

    private static final String URL = "jdbc:mysql://localhost:3306/orderm";
    private static final String USER = "root";
    private static final String PASSWORD = "fahi";

    public static Connection getDBConn() throws SQLException {
        try {
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            System.out.println("❌ Database connection failed: " + e.getMessage());
            throw e;
        }
    }
```

```
Javadoc  Declaration  Console ×  Install Java 24 Support
rminated> DBUtil (1) [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
Connected successfully!
```

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".

```
package com.main;

import com.product.Product;
import com.user.User;
import com.user.Order;
import com.service.OrderProcessor;
import com.exception.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class OrderManagementApp {

    public static void main(String[] args) {


        List<User> users = new ArrayList<>();
        List<Product> products = new ArrayList<>();
        List<Order> orders = new ArrayList<>();


        OrderProcessor orderProcessor = new OrderProcessor(users, products, orders);


        users.add(new User(1, "Alice", false));
        users.add(new User(2, "Admin", true));

        products.add(new Product(1, "Laptop", 1500));
        products.add(new Product(2, "Smartphone", 800));


        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\nMenu:");
            System.out.println("1. Create User");
```

```
        System.out.println("2. Create Product");
        System.out.println("3. Cancel Order");
        System.out.println("4. Get All Products");
        System.out.println("5. Get Orders by User");
        System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:

                System.out.print("Enter User ID: ");
                int userId = scanner.nextInt();
                scanner.nextLine();
                System.out.print("Enter User Name: ");
                String userName = scanner.nextLine();
                System.out.print("Is Admin (true/false): ");
                boolean isAdmin = scanner.nextBoolean();
                User newUser = new User(userId, userName, isAdmin);
                try {
                    orderProcessor.createUser(newUser);
                } catch (UserAlreadyExistsException e) {
                    System.out.println(e.getMessage());
                }
                break;

            case 2:

                System.out.print("Enter Admin User ID: ");
                int adminId = scanner.nextInt();
                scanner.nextLine();
                System.out.print("Enter Product ID: ");
                int productId = scanner.nextInt();
                scanner.nextLine();
                System.out.print("Enter Product Name: ");
                String productName = scanner.nextLine();
                System.out.print("Enter Product Price: ");
```

```
            double productPrice = scanner.nextDouble();
            Product newProduct = new Product(productId, productName, productPrice);
            User adminUser = users.stream()
                    .filter(u -> u.getId() == adminId)
                    .findFirst()
                    .orElse(null);
            try {
                if (adminUser != null && adminUser.isAdmin()) {
                    orderProcessor.createProduct(adminUser, newProduct);
                } else {
                    System.out.println("You are not an admin!");
                }
            } catch (AdminNotFoundException e) {
                System.out.println(e.getMessage());
            }
            break;

        case 3:

            System.out.print("Enter User ID: ");
            int cancelUserId = scanner.nextInt();
            System.out.print("Enter Order ID to cancel: ");
            int orderId = scanner.nextInt();
            try {
                orderProcessor.cancelOrder(cancelUserId, orderId);
            } catch (UserNotFoundException | OrderManagementException e) {
                System.out.println(e.getMessage());
            }
            break;

        case 4:

            List<Product> allProducts = orderProcessor.getAllProducts();
            System.out.println("Products available:");
            for (Product product : allProducts) {
                System.out.println(product);
            }
            break;

         case 5:

            System.out.print("Enter User ID: ");
            int orderUserId = scanner.nextInt();
            scanner.nextLine();
            User userOrders = users.stream()
                    .filter(u -> u.getId() == orderUserId)
                    .findFirst()
                    .orElse(null);
            if (userOrders != null) {
                try {
                    List<Order> userOrdersList = orderProcessor.getOrderByUser(userOrders);
                    if (userOrdersList.isEmpty()) {
                        System.out.println("No orders found for this user.");
                    } else {
                        System.out.println("Orders for User " + userOrders.getName() + ":");
                        for (Order order : userOrdersList) {
                            System.out.println(order);
                        }
                    }
                } catch (UserNotFoundException e) {
                    System.out.println(e.getMessage());
                }
            } else {
                System.out.println("User not found!");
            }
            break;

         case 6:
            // Exit
            System.out.println("Exiting the system...");
            break;

         default:
            System.out.println("Invalid choice! Please try again.");
            break;
    }
} while (choice != 6);
```

```
153         } while (choice != 6);
154
155         scanner.close();
156     }
157 }
158
```

Javadoc | Declaration | Console | Install Java 24 Support

OrderManagementApp [Java Application] C:\Users\nisha\.p2\pool\plugins\org.eclipse.justj.ope

```
Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
Enter your choice: 1
Enter User ID: 1
Enter User Name: sam
Is Admin (true/false): true
User with ID 1 already exists.

Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
Enter your choice:
```