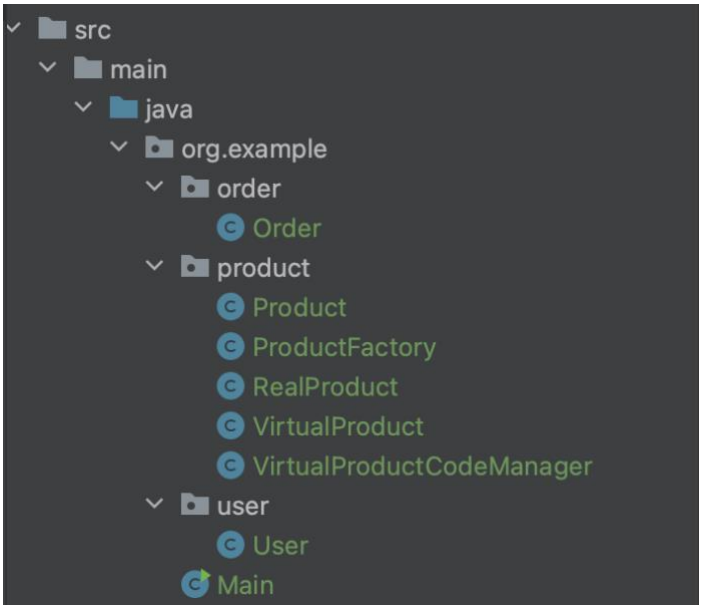
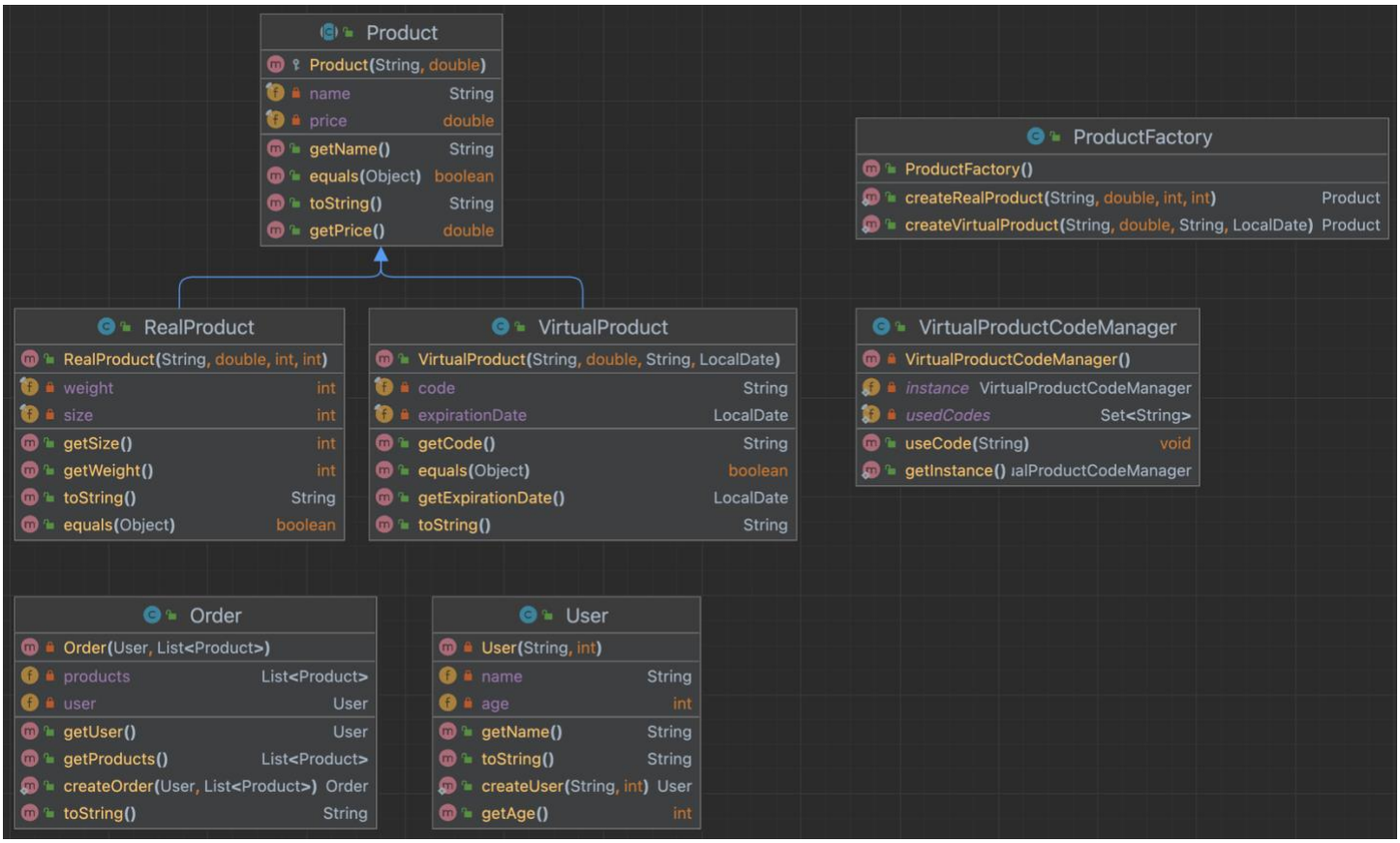
Test Assignment Project Structure:



Separation of classes by packages is made just for cleanness purposes (instead of writing the entire code of the solution in just one file).

Project Class Diagram:



Product is a base abstract class representing a product. The *Product* class is made abstract intentionally since we can divide every product possible in the world into two groups: real products (e.g. a car, a house, a laptop) and virtual products (e.g. some paid software). Taking this fact into account we can conclude that there will not be need for creating instances of the *Product* class itself. Generally, a “real product” and a “virtual product” are at the highest level of abstraction where we can already create product instances of these concrete implementations. The *Product* class just dictates common properties and functionality that both real products and virtual products have.
ProductFactory is a factory class used to create real and virtual product objects.
VirtualProductCodeManager is a classes used to use virtual product codes and check whether some particular code was already used or not.
Order and *User* classes represent orders and users respectively.

Source Code of the Solution:

User.java:

```java
package org.example.user;

public class User {
    private String name;
    private int age;

    private User(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public static User createUser(String name, int age) {
        return new User(name, age);
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

Product.java:

```java
package org.example.product;

public abstract class Product {
    private final String name;
    private final double price;

    protected Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Product product)) return false;

        if (Double.compare(product.getPrice(), getPrice()) != 0) return false;
        return getName().equals(product.getName());
    }

    @Override
    public String toString() {
        return "Product{" +
                "name='" + name + '\'' +
                ", price=" + price +
                '}';
    }
}
```

VirtualProduct.java:

```
package org.example.product;

import java.time.LocalDate;

public class VirtualProduct extends Product {
    private final String code;
    private final LocalDate expirationDate;

    public VirtualProduct(String name, double price, String code, LocalDate
expirationDate) {
        super(name, price);
        this.code = code;
        this.expirationDate = expirationDate;
    }

    public String getCode() {
        return code;
    }

    public LocalDate getExpirationDate() {
        return expirationDate;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof VirtualProduct that)) return false;
        if (!super.equals(o)) return false;

        if (getCode() != null ? !getCode().equals(that.getCode()) : that.getCode() !=
null) return false;
        return getExpirationDate() != null ?
getExpirationDate().equals(that.getExpirationDate()) : that.getExpirationDate() == null;
    }

    @Override
    public String toString() {
        return "VirtualProduct{" +
                "name='" + getName() + '\'' +
                ", price=" + getPrice() +
                ", code='" + code + '\'' +
                ", expirationDate=" + expirationDate +
                '}';
    }
}
```

RealProduct.java:

```java
package org.example.product;

public class RealProduct extends Product {
    private final int size;
    private final int weight;

    public RealProduct(String name, double price, int size, int weight) {
        super(name, price);
        this.size = size;
        this.weight = weight;
    }

    public int getSize() {
        return size;
    }

    public int getWeight() {
        return weight;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof RealProduct that)) return false;
        if (!super.equals(o)) return false;

        if (getSize() != that.getSize()) return false;
        return getWeight() == that.getWeight();
    }

    @Override
    public String toString() {
        return "RealProduct{" +
                "name='" + getName() + '\'' +
                ", price=" + getPrice() +
                ", size=" + size +
                ", weight=" + weight +
                '}';
    }
}
```

ProductFactory.java:

```
package org.example.product;

import java.time.LocalDate;

public class ProductFactory {
    public static Product createRealProduct(String name, double price, int size, int
weight) {
        return new RealProduct(name, price, size, weight);
    }

    public static Product createVirtualProduct(String name, double price, String code,
LocalDate expirationDate) {
        return new VirtualProduct(name, price, code, expirationDate);
    }
}
```

Order.java:

```java
package org.example.order;

import org.example.user.User;
import org.example.product.Product;

import java.util.List;

public class Order {
    private User user;
    private List<Product> products;

    private Order(User user, List<Product> products) {
        this.user = user;
        this.products = products;
    }

    public static Order createOrder(User user, List<Product> products) {
        return new Order(user, products);
    }

    public User getUser() {
        return user;
    }

    public List<Product> getProducts() {
        return products;
    }

    @Override
    public String toString() {
        return "Order{" +
                "user=" + user +
                ", products=" + products +
                '}';
    }
}
```

VirtualProductCodeManager.java (it’s a Singleton class):

```java
package org.example.product;

import java.util.HashSet;
import java.util.Set;

public class VirtualProductCodeManager {
    private static VirtualProductCodeManager instance;
    private static final Set<String> usedCodes = new HashSet<>();

    private VirtualProductCodeManager() {

    }

    public static VirtualProductCodeManager getInstance() {
        if (instance == null) {
            instance = new VirtualProductCodeManager();
        }
        return instance;
    }

    public void useCode(String code) {
        usedCodes.add(code);
    }

    public boolean isCodeUsed(String code) {
        return usedCodes.contains(code);
    }
}
```

Main.java:

```java
package org.example;

import org.example.order.Order;
import org.example.product.Product;
import org.example.product.ProductFactory;
import org.example.product.RealProduct;
import org.example.product.VirtualProductCodeManager;
import org.example.user.User;

import java.time.LocalDate;
import java.util.*;

public class Main {
    public static void main(String[] args) {

        //TODO Create User class with method createUser
        // User class fields: name, age;
        // Notice that we can only create user with createUser method without using
constructor or builder
        User user1 = User.createUser("Alice", 32);
        User user2 = User.createUser("Bob", 19);
        User user3 = User.createUser("Charlie", 20);
        User user4 = User.createUser("John", 27);

        //TODO Create factory that can create a product for a specific type: Real or
Virtual
        // Product class fields: name, price
        // Product Real class additional fields: size, weight
        // Product Virtual class additional fields: code, expiration date

        Product realProduct1 = ProductFactory.createRealProduct("Product A", 20.50, 10,
25);
        Product realProduct2 = ProductFactory.createRealProduct("Product B", 50, 6, 17);

        Product virtualProduct1 = ProductFactory.createVirtualProduct("Product C", 100,
"xxx", LocalDate.of(2023, 5, 12));
        Product virtualProduct2 = ProductFactory.createVirtualProduct("Product D", 81.25,
"yyy",  LocalDate.of(2024, 6, 20));


        //TODO Create Order class with method createOrder
        // Order class fields: User, List<Price>
        // Notice that we can only create order with createOrder method without using
constructor or builder
        List<Order> orders = new ArrayList<>() {{
            add(Order.createOrder(user1, List.of(realProduct1, virtualProduct1,
virtualProduct2)));
            add(Order.createOrder(user2, List.of(realProduct1, realProduct2)));
            add(Order.createOrder(user3, List.of(realProduct1, virtualProduct2)));
            add(Order.createOrder(user4, List.of(virtualProduct1, virtualProduct2,
realProduct1, realProduct2)));
        }};

        //TODO 1). Create singleton class which will check the code is used already or
not
        // Singleton class should have the possibility to mark code as used and check if
code used
        // Example:
        // singletonClass.useCode("xxx")
        // boolean isCodeUsed = virtualProductCodeManager.isCodeUsed("xxx") --> true;
        // boolean isCodeUsed = virtualProductCodeManager.isCodeUsed("yyy") --> false;
```

```
        System.out.println("1. Create singleton class VirtualProductCodeManager \n");
        VirtualProductCodeManager codeManager = VirtualProductCodeManager.getInstance();
// Added by Serhii Chernikov
        codeManager.useCode("xxx");
// Added by Serhii Chernikov
        boolean isXxxUsed = codeManager.isCodeUsed("xxx"); // should be set to 'true'
        System.out.println("Is code 'xxx' used: " + isXxxUsed + "\n");
        boolean isYyyUsed = codeManager.isCodeUsed("yyy"); // should be set to 'false'
// Added by Serhii Chernikov
        System.out.println("Is code 'yyy' used: " + isYyyUsed + "\n");
// Added by Serhii Chernikov

        //TODO 2). Create a functionality to get the most expensive ordered product
        Product mostExpensive = getMostExpensiveProduct(orders);
        System.out.println("2. Most expensive product: " + mostExpensive + "\n");

        //TODO 3). Create a functionality to get the most popular product(product bought
by most users) among users
        Product mostPopular = getMostPopularProduct(orders);
        System.out.println("3. Most popular product: " + mostPopular + "\n");

        //TODO 4). Create a functionality to get average age of users who bought
realProduct2
        double averageAge = calculateAverageAge(realProduct2, orders);
        System.out.println("4. Average age is: " + averageAge + "\n");

        //TODO 5). Create a functionality to return map with products as keys and a list
of users
        // who ordered each product as values
        Map<Product, List<User>> productUserMap = getProductUserMap(orders);
        System.out.println("5. Map with products as keys and list of users as value \n");
        productUserMap.forEach((key, value) -> System.out.println("key: " + key + " " +
"value: " +  value + "\n"));

        //TODO 6). Create a functionality to sort/group entities:
        // a) Sort Products by price
        // b) Sort Orders by user age in descending order
        List<Product> productsByPrice = sortProductsByPrice(List.of(realProduct1,
realProduct2, virtualProduct1, virtualProduct2));
        System.out.println("6. a) List of products sorted by price: " + productsByPrice +
"\n");
        List<Order> ordersByUserAgeDesc = sortOrdersByUserAgeDesc(orders);
        System.out.println("6. b) List of orders sorted by user agge in descending order:
" + ordersByUserAgeDesc + "\n");

        //TODO 7). Calculate the total weight of each order
        Map<Order, Integer> result = calculateWeightOfEachOrder(orders);
        System.out.println("7. Calculate the total weight of each order \n");
        result.forEach((key, value) -> System.out.println("order: " + key + " " + "total
weight: " +  value + "\n"));
    }

    private static Product getMostExpensiveProduct(List<Order> orders) {
        Product dearestProduct = null;
        double highestPrice = Double.MIN_VALUE;

        for (Order order : orders) {
            for (Product product : order.getProducts()) {
                if (product.getPrice() > highestPrice) {
                    dearestProduct = product;
                    highestPrice = product.getPrice();
                }
            }
```

```
        }

        return dearestProduct;
    }

    private static Product getMostPopularProduct(List<Order> orders) {
        Map<Product, Integer> productPurchaseCount = populateMapOfPurchaseCounts(orders);

        Product mostPopularProduct = null;
        int purchaseCount = 0;

        for (Map.Entry<Product, Integer> entry : productPurchaseCount.entrySet()) {
            if (entry.getValue() > purchaseCount) {
                mostPopularProduct = entry.getKey();
                purchaseCount = entry.getValue();
            }
        }

        return mostPopularProduct;
    }

    /**
     * A helper method that populates the Map with pairs of types Product as a key, and
Integer as a value.
     * The integer value represents the number of times some particular product was
purchased.
     */
    private static Map<Product, Integer> populateMapOfPurchaseCounts(List<Order> orders)
{
        Map<Product, Integer> productPurchaseCount = new HashMap<>();

        for (Order order : orders) {
            for (Product product : order.getProducts()) {
                if (productPurchaseCount.containsKey(product)) {
                    productPurchaseCount.replace(product,
productPurchaseCount.get(product)+1);
                } else {
                    productPurchaseCount.put(product, 1);
                }
            }
        }

        return productPurchaseCount;
    }

    private static double calculateAverageAge(Product product, List<Order> orders) {
        double ageSum = 0;
        int usersCount = 0;

        for (Order order : orders) {
            for (Product prdct : order.getProducts()) {
                if (prdct.equals(product)) {
                    ageSum += order.getUser().getAge();
                    usersCount++;
                }
            }
        }

        return ageSum / usersCount;
    }

    private static Map<Product, List<User>> getProductUserMap(List<Order> orders) {
        Map<Product, List<User>> productToUsers = new HashMap<>();
```

```
        for (Order order : orders) {
            for (Product product : order.getProducts()) {
                if (productToUsers.containsKey(product)) {
                    productToUsers.get(product).add(order.getUser());
                } else {
                    List<User> users = new ArrayList<>();
                    users.add(order.getUser());
                    productToUsers.put(product, users);
                }
            }
        }

        return productToUsers;
    }

    private static List<Product> sortProductsByPrice(List<Product> products) {
        return
products.stream().sorted(Comparator.comparing(Product::getPrice)).toList();
    }

    private static List<Order> sortOrdersByUserAgeDesc(List<Order> orders) {
        Comparator<Order> priceComparator = Comparator.comparing(order ->
order.getUser().getAge());
        orders.sort(priceComparator.reversed());
        return orders;
    }

    private static Map<Order, Integer> calculateWeightOfEachOrder(List<Order> orders) {
        Map<Order, Integer> orderWeights = new HashMap<>();

        for (Order order : orders) {
            int totalWeightOfCurrentOrder = 0;
            for (Product product : order.getProducts()) {
                if (product instanceof RealProduct) {
                    totalWeightOfCurrentOrder += ((RealProduct) product).getWeight();
                }
            }
            orderWeights.put(order, totalWeightOfCurrentOrder);
        }

        return orderWeights;
    }
}
```

The Console Output After Running the main() Method:

```
1. Create singleton class VirtualProductCodeManager

Is code 'xxx' used: true

Is code 'yyy' used: false

2. Most expensive product: VirtualProduct{name='Product C', price=100.0, code='xxx', expirationDate=2023-05-12}

3. Most popular product: RealProduct{name='Product A', price=20.5, size=10, weight=25}

4. Average age is: 23.0

5. Map with products as keys and list of users as value

key: VirtualProduct{name='Product D', price=81.25, code='yyy', expirationDate=2024-06-20} value: [User{name='Alice', age=32}, User{name='Charlie', age=20}, User{name='John', age=27}]

key: RealProduct{name='Product A', price=20.5, size=10, weight=25} value: [User{name='Alice', age=32}, User{name='Bob', age=19}, User{name='Charlie', age=20}, User{name='John', age=27}]

key: VirtualProduct{name='Product C', price=100.0, code='xxx', expirationDate=2023-05-12} value: [User{name='Alice', age=32}, User{name='John', age=27}]

key: RealProduct{name='Product B', price=50.0, size=6, weight=17} value: [User{name='Bob', age=19}, User{name='John', age=27}]

6. a) List of products sorted by price: [RealProduct{name='Product A', price=20.5, size=10, weight=25}, RealProduct{name='Product B', price=50.0, size=6, weight=17}, VirtualProduct{name='Product

6. b) List of orders sorted by user agge in descending order: [Order{user=User{name='Alice', age=32}, products=[RealProduct{name='Product A', price=20.5, size=10, weight=25}, VirtualProduct{name=

7. Calculate the total weight of each order

order: Order{user=User{name='Bob', age=19}, products=[RealProduct{name='Product A', price=20.5, size=10, weight=25}, RealProduct{name='Product B', price=50.0, size=6, weight=17}]} total weight: 4

order: Order{user=User{name='Alice', age=32}, products=[RealProduct{name='Product A', price=20.5, size=10, weight=25}, VirtualProduct{name='Product C', price=100.0, code='xxx', expirationDate=20

order: Order{user=User{name='Charlie', age=20}, products=[RealProduct{name='Product A', price=20.5, size=10, weight=25}, VirtualProduct{name='Product D', price=81.25, code='yyy', expirationDate=

order: Order{user=User{name='John', age=27}, products=[VirtualProduct{name='Product C', price=100.0, code='xxx', expirationDate=2023-05-12}, VirtualProduct{name='Product D', price=81.25, code='y
```