UNIVERSITÉ DU LUXEMBOURG

Programming 1

Ali Osman TOPAL (aliosman.topal@uni.lu)

Lab 8 – Generics, enum & equals/hashCode

2021–11–12

Submission (Preliminary: 21/11, Final: 28/11)
The mandatory exercises for **Code Submission** are **1, 2, and 4** (marked with an ❗). All other exercises on this sheet are optional but still highly recommended! The **Explainer Video** 🎥 for this sheet must be realized on **Exercise 1 ("Package Delivery")**. A Flipgrid invitation link will be posted on Moodle.

Exercise 1 – ❗ 🎥 Package Delivery #enum #Class

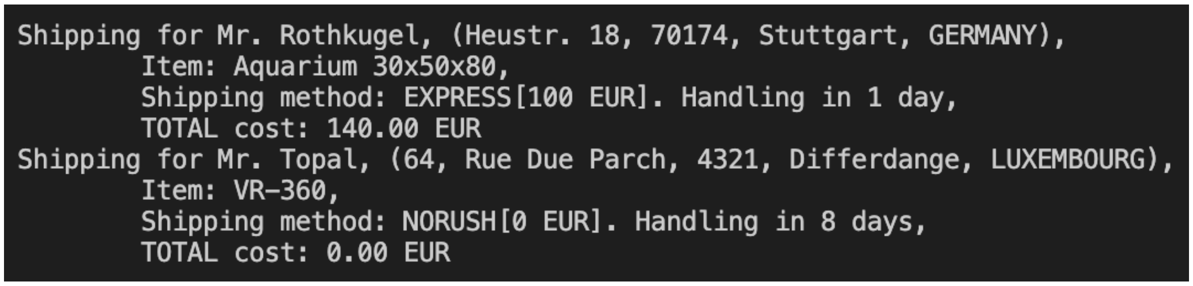
You are working at a package transport company to digitise the old system. Packages can be shipped to recipients living in one of four countries: Luxembourg, France, Belgium and Germany. Four different shipping methods are offered, with different handling times: EXPRESS (1 day), PRIORITY (3 days), CLASSIC (5 days), and NORUSH (8 days).

The shipping cost is composed of a country-specific base price, plus a fee depending on the shipping method. The base price for the different countries is as follows: France 30 EUR, Belgium 50 EUR, Germany 40 EUR, and Luxembourg is free of charge. The shipping methods' fee is 100 EUR for EXPRESS, 50 EUR for PRIORITY, 10 EUR for CLASSIC, and free of charge for NORUSH. For example, if a package is sent by EXPRESS within Luxembourg, the total cost is $100 + 0 = 100$ EUR. For reasons of simplicity, neither the size nor the weight of the package are relevant for the total cost.

When a package is ready to transport, you want your system to print the following data: name and address of the recipient, the item shipped, the shipping method together with its fee and handling time, and the total cost including the base price.

Hint:
Use `enum` to represent a discrete set of values. Keep in you mind that you can associate *multiple values* with each `enum` constant. For more information, you might want to consult the scrapbook, which refers to the Core Java book, in addition to supplementary sources such as the Java tutorial available at:
https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html.

Finally, write a main program to test your implementation. An example of the output is shown below.



```
Shipping for Mr. Rothkugel, (Heustr. 18, 70174, Stuttgart, GERMANY),
        Item: Aquarium 30x50x80,
        Shipping method: EXPRESS[100 EUR]. Handling in 1 day,
        TOTAL cost: 140.00 EUR
Shipping for Mr. Topal, (64, Rue Due Parch, 4321, Differdange, LUXEMBOURG),
        Item: VR-360,
        Shipping method: NORUSH[0 EUR]. Handling in 8 days,
        TOTAL cost: 0.00 EUR
```

Figure 1: A sample output for Exercise 1

Programming 1

Ali Osman TOPAL (aliosman.topal@uni.lu)

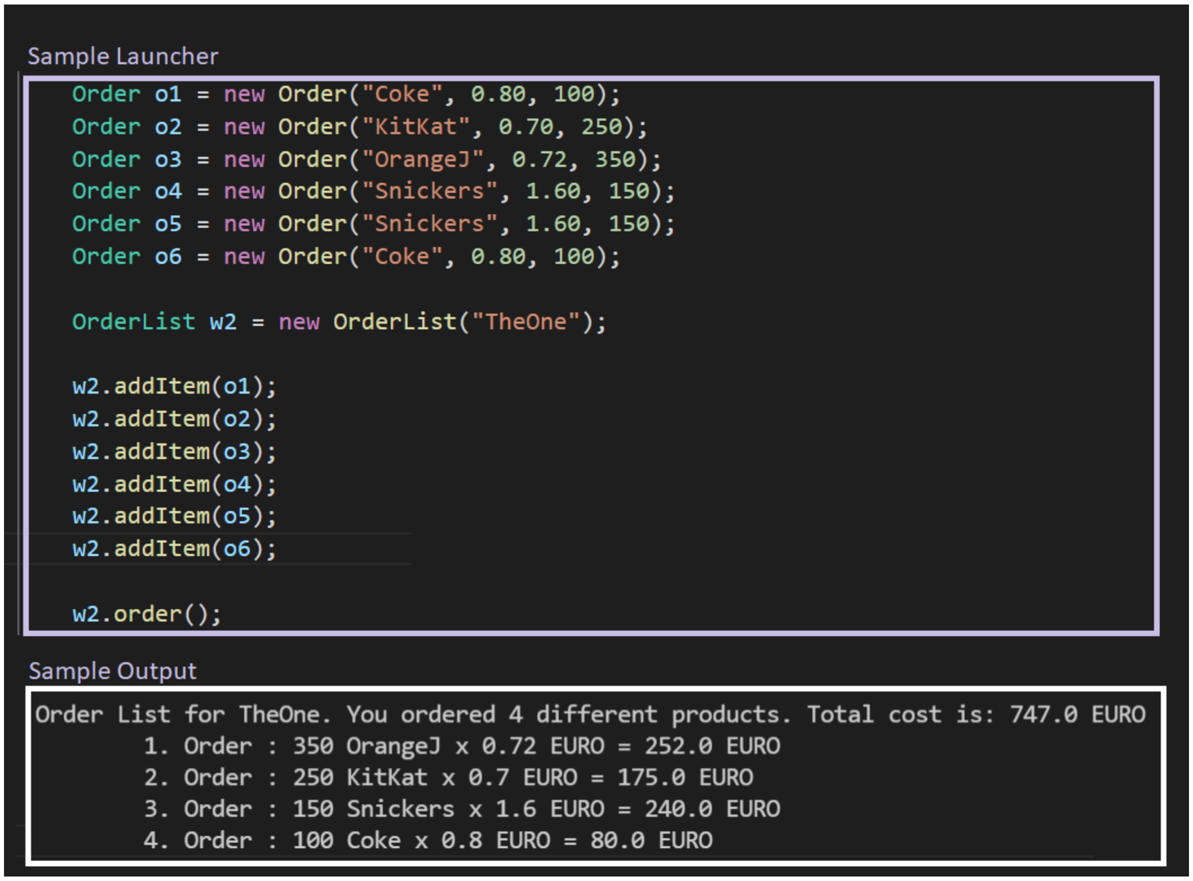
Lab 8 – Generics, enum & equals/hashCode

Exercise 2 – ❗ Store #equals #hashCode

Your uncle has been running a big store for 15 years. He receives the requests from the small markets, prepares the order list and sends it to the sales manager for delivery. However, due to age, he has begun to mix up the orders. Sometimes he enters the same item repeatedly, at any time, as illustrated in the sample launcher below. Help your uncle with a software detecting and avoiding duplicate items.

Every order item is composed of a product name, a unit price, and an amount. An order list in turn contains the name of the client store, together with all items that this client ordered. To detect and prevent duplicates, use a `HashSet`, together with proper implementations of `equals` and `hashCode()`. An order item is considered a duplicate if all its properties match, i.e., the product name, unit price, and amount are the same. Please note that no particular sorting order is required, i.e., the items can appear in an arbitrary order when printing the list of items.

Design and implement the corresponding classes, and write a main program to test your implementation. The output of your program should follow the sample from Figure 2 below:



Sample Launcher

```java
Order o1 = new Order("Coke", 0.80, 100);
Order o2 = new Order("KitKat", 0.70, 250);
Order o3 = new Order("OrangeJ", 0.72, 350);
Order o4 = new Order("Snickers", 1.60, 150);
Order o5 = new Order("Snickers", 1.60, 150);
Order o6 = new Order("Coke", 0.80, 100);

OrderList w2 = new OrderList("TheOne");

w2.addItem(o1);
w2.addItem(o2);
w2.addItem(o3);
w2.addItem(o4);
w2.addItem(o5);
w2.addItem(o6);

w2.order();
```

Sample Output

```
Order List for TheOne. You ordered 4 different products. Total cost is: 747.0 EURO
        1. Order : 350 OrangeJ x 0.72 EURO = 252.0 EURO
        2. Order : 250 KitKat x 0.7 EURO = 175.0 EURO
        3. Order : 150 Snickers x 1.6 EURO = 240.0 EURO
        4. Order : 100 Coke x 0.8 EURO = 80.0 EURO
```

Figure 2: A sample order and the output for Exercise 2

Exercise 3 – `hashCode()` Performance Tests #equals

We have seen in the lecture that whenever two objects are considered equal, the contract of the `hashCode` methods states that the returned integer value also needs to be equal. Though it is not required, objects that are not considered equal should have a differing hash code to improve the performance of a certain number of operations in hash tables.

The purpose of this exercise is to show that a proper implementation of `hashCode` can have a significant impact on the performance of operations such as adding and retrieving elements in a `HashSet`.

Define a class with some attributes. Make sure the `equals` method is properly implemented. Provide different implementations of the `hashCode` method, ranging from a constant integer value returned to highly dynamic values.

Investigate the performance impact with respect to the differentiation of the returned hash values for both filling and retrieving elements in a vast hash set.

Variate your experiments for different numbers of elements in the set and for different implementations of `hashCode`. Document your observations.

Exercise 4 – ❗ Supermarket Loyalty Card #GenericType #GenericParameter #equals

A supermarket gives loyalty cards to its customers. Every card has an ID and a number of accumulated loyalty points. Each time a customer buys at the supermarket, 10% of the total amount are added as points to the card. It is also possible to redeem an amount by paying with the points on the loyalty card.

The supermarket gives several cards to a customer, which can be used by her family members. The ID of the different cards of a same customer account are equal. Two loyalty cards are considered equal if their IDs are the same. For simplicity, you can assume that accumulated points are not shared among the different cards of a customer.

The supermarket holds the information of all cards that have redeemed on a certain date. For that, you might use a data structure such as `Map<LocalDate, List<LoyaltyCard>>`, which maps a date to a list of loyalty cards (the current day can be determined with `LocalDate.now()`).

A redeem action is only successful if a card has not yet been used to redeem on the same day and if the points of the individual card are sufficient for paying the amount due.