# Training materials

* The Java Tutorials. [Lesson: Generics](https://docs.oracle.com/javase/tutorial/java/generics/index.html).
* The Java Tutorials. [Lesson: Exceptions](https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html).
* The Java Tutorials. [Trail: Collections](https://docs.oracle.com/javase/tutorial/collections/index.html).

# Code Exercise

Create the package named by.epam.inner for immutable classes of entities.

1. Byn - a financial value for BYN (belarusian rubles).

String representation of a financial value has the format:

d+.dd

1. Product with two fields: name and price in BYN.

String representation of a product has the csv format:

name;price

1. Superclass Purchase that represents a product purchase. It has two fields: product and number of purchased units.

String representation of a purchase has the csv format:

product;number;cost

where cost is a purchase cost in Byn, and cost = price \* number.

1. Subclass PriceDiscountPurchase that represents a product purchase with a price discount. It has one field: price discount.

String representation of a purchase has the csv format:

product;number;discount;cost

where cost = (price - discount) \* number.

Any two purchases are equal if their products are equal.

Any two products are equal if their names and prices are equal.

A csv file contains a series of text lines. Every line contains information about a purchase date and a purchase itself in the format:

date;name;price;number

or

date;name;price;number;discount

A date format: yy-mm-dd, where yy - two last digits of a year of this century, mm is for month and dd is for day.

A name format: any non empty string.

A price and a discount format: d+.dd, but not 0.00.

A number format: d+, but not 0.

Wrong lines in a csv file are possible. Log wrong lines during the process of loading a content of a csv file with the help of Log4j api. A record must contain a wrong line itself and the reason(s) of error.

Define the Runner class in the default package. An argument of the Runner is the name of a csv file.

The algorithm of the method main( ):

1. load the content of a csv file into **the map** where a purchase is a key and a date of a last purchase in a csv file is a value;
2. load the content of a csv file into **the map** where a purchase is a key and a date of a first purchase in a csv file is a value;
3. load the content of a csv file into **the enumerated map** where a weekday is a key and purchases list for this weekday is a value;
4. load instances of the subclass PriceDiscountPurchase from a csv file into **the** **List**<**PriceDiscountPurchase**> **discountPurchases**;
5. output maps to the console;
6. find and output the dates for bread with price 1.55 in the second map;
7. find and output the date for bread with price 1.70 in the first map;
8. find and output all purchases on MONDAY;
9. output the total cost of all purchases for each weekday;
10. output the total cost of purchases from the list **discountPurchases**;
11. remove all entries from the first map where the purchase name is meat;
12. remove all entries from the second map on FRIDAY;
13. remove all entries from the second map where the cost is less than   
    10.00 BYN and year is 2018;
14. remove all entries from the enumerated map where a product name is milk in some purchase;
15. output maps to the console.

The example of a csv file:

19-02-15;bread;1.55;1;0.20

20-01-03;milk;1.31;2

19-02-15;bread;1.50

20-03-24;bread;1.54;3

18-11-30;bread;1.55;5

11-30;bread;1.55;5

potato;0.80;2;0.10

20-02-10;potato;0.80;2;0.10

90-02-15;bread;1.50;1

20-03-24;butter;2.70;1

20-03-24;butter;2.41;1;0.50

19-02-15;bread;1.50;1;1

19-20-15;bread;2.50;2

19-20-15;bread;2.50;-2

20-01-03;meat;8.00;2;0.90

18-11-30;potato;0.80;5;0.10

19-12-15;milk;1.31;2

20-01-03;bread;1.54;3

20-05-10;bread;1.55;5

20-04-12;meat;9.00;2;0.70

20-01-03;potato;0.80;3;0.20

Требования к проекту

– Использовать фреймворк maven.

– Покрыть код junit тестами.

Ограничения и замечания к задаче 1

– Данная задача предназначена для повторения параметризованных типов, исключений и коллекций.

– **Запрещен** функционал **java 8** (стримы, лямбды, появившиеся классы и интерфейсы).

– **Запрещены** [raw types](https://docs.oracle.com/javase/tutorial/java/generics/rawTypes.html). Как следствие, не использовать класс Object. Это задача как по теме [Collections framework](https://docs.oracle.com/javase/tutorial/collections/index.html), так и [**Generic Types and Methods**](https://docs.oracle.com/javase/tutorial/java/generics/index.html) **(!!!)**.

– В данной задаче на все финансовые величины ввожу следующее ограничение: **максимальное значение стоимости в копейках не больше 0x7fffffff** (т.е. максимум диапазона int). Следствие: выход за пределы диапазона не контролировать.

– Запрещаю использовать функционал класса BigDecimal.

– Читайте у Джошуа Блоха раздел "Если требуются точные ответы, избегайте использования типов float и double". Следствие: **вещественный (плавающий) тип не подходит для финансовых расчетов**. Вот еще один убедительный аргумент - ссылка на статью [Representing money](http://www.javapractices.com/topic/TopicAction.do?Id=13) от авторов отличного ресурса [Java practices](http://www.javapractices.com/home/HomeAction.do).

– Создать три мэпа (пункты 1, 2, 3) и лист (пункт 4) за **один** проход по файлу (в одном цикле и без дополнительных структур данных).

– Критерий дня первой/последней (first/last) покупки зависит от очередности появления **в файле**, а не по правилу первая покупка - это та покупка, которая самая ранняя, а последняя - та, которая самая поздняя.

– Создать **единый** приватный статический метод раннера для вывода мэпов на консоль. Формат вывода - последовательность строк вида key => value, см. примеры ниже.

– Создать **единый** приватный статический метод раннера для поиска элемента в мэпе (пункты 6, 7, 8).

– Для вычисления суммы покупок по листу создать **единый** статический метод раннера и использовать его для пунктов 9 и 10.

– Создать **единый** приватный статический метод раннера для удаления элементов из мэпа (пункты 11 - 14).

– Не забыть проверить решение, когда нет исходного файла.

– После чтения вышеприведенного файла должны быть получены следующие мэпы:

sorry, coming soon...

– Этапы выполнения задачи 1

Этап 1. Реализация нулевой версии решения.

– Заполните [форму регистрации дедлайна](https://docs.google.com/forms/d/e/1FAIpQLSe2ph0LJpVgTGXaE9_xLnbucIFF4MORP34HwdG5g1GYRfcYNg/viewform) первой версии.

– Создайте архив с исходниками решения **start1.zip**.

На всякий случай откройте архив и убедитесь, что в нем находятся только исходники и ресурсы. В крайнем случае, убирайте руками из архива файлы от IDE.

– Выложите архив в свою папку.  
Этап 2. Методические рекомендации по решению.

– После появления jar-файла скрипт откроет доступ (в 4:00+) к текстовому документу, где описан процесс реализации задачи.

– Исправьте присутствующие в Вашем решении недостатки.

– Обсудите свое решение с трэйни, который уже выполнил задачу start1.

– Заполните [форму рецензирования](https://docs.google.com/forms/d/e/1FAIpQLSf6jW2oiE7AiPTZAS2cOUbFj76rEPW7z-WS-D_Xaq6_wKnPcg/viewform), указав в поле **Result** значение **yes**.

Этап 3. Контроль.

– Исправьте отмеченные недостатки и выложите обновленное решение в архиве **start1\_2.zip** в свою папку.

– Скрипт заберет решение.

– Тренер выложит рецензию в документ **notes** в вашей папке.

– Если рецензия отрицательная, то повторите этапы 2 и 3, но с другим рецензентом.

– Если рецензия положительная, то выполняйте следующее задание.