# Training materials

* The Java Tutorials. [Lambda Expressions](https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html).
* The Java Tutorials. [Method References](https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html).
* The Java Tutorials. [The Collection Interface](https://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html). Subsection Aggregate Operations.
* The Java Tutorials. [Lesson: Aggregate Operations](https://docs.oracle.com/javase/tutorial/collections/streams/index.html).
* The Java Tutorials. [Reduction](https://docs.oracle.com/javase/tutorial/collections/streams/reduction.html).
* [Consider Using Java SE 8's Optional! - Oracle](https://www.oracle.com/technical-resources/articles/java/java8-optional.html).

# Code Exercise

Create version 2 of the task start1 using java 8 new features.

New format of a csv file line:

date;class;args

date - the same.

class - the name of a purchase superclass or subclass

args - arguments of a constructor in the csv format.

Examples of valid lines:

19-05-15;Purchase;milk;1.70;3

20-04-01;PriceDiscountPurchase;bread;1.20;3;0.10

20-06-20;PercentDiscountPurchase;cheese;5.00;20;5.825

The superclass Purchase and the subclass PriceDiscountPurchase are the same as in the start1 project. Class names may differ.

Define the subclass PercentDiscountPurchase (or some other name) for the purchase with a percent discount to be presented if the number of purchased units is greater than the given subclass constant. A discount rate is given by the percentage from the purchase cost. Override necessary methods.

Требования к проекту

– Использовать фреймворк maven.

– Покрыть код junit тестами.

Ограничения и замечания к задаче start2

– Данная задача предназначена для повторения функционала **java 8** (стримы, лямбды, появившиеся классы и интерфейсы).

– **Запрещено** использовать пользовательские интерфейсы (EntryChecker и т.п.).

– **Рефлексию** для преобразования из строки в имя класса **не использовать**. Задействовать программный интерфейс перечислений.

– From J. Bloch, 3-rd edition, item 55, p. 250.

"So how do you choose to return an optional instead of throwing an exception? **Optionals are similar in spirit to checked exceptions** (Item 71), in that they force the user of an API to confront the fact that there may be no value returned. Throwing an unchecked exception or returning a null allows the user to ignore this eventuality, with potentially dire consequences. However, throwing a checked exception requires additional boilerplate code in the client".

Следствие. **Запрещено** создавать обрабатываемые (checked) исключения, использовать вместо них функционал Optional.

– Примеры расчета стоимости для второго подкласса при условии количества, которое надо превысить, равного 15 (константа класса).

пример 1

20-06-20;PercentDiscountPurchase;cheese;5.00;20;5.825

т.к. 20 > 15, то стоимость = 5.00 \* 20 \* (1 – 5.825/100) = 94.18.

пример 2

20-06-20;PercentDiscountPurchase;cheese;5.00;10;5.825

т.к. 10 <= 15, то стоимость = 5.00 \* 20 = 100.00.

# Guidelines

**Ключевая идея**

Главное отличие версии 2 задачи заключается в использовании функционала класса Optional вместо обрабатываемых (checked) исключений.

Цикл чтения файла и создания мэпов и листа в версии 1 схематически имеет вид:

try(Scanner sc = new Scanner(new FileReader(args[0]));) {

while(sc.hasNext()) {

try {

String line = sc.nextLine();

WrapperEntry entry = new WrapperEntry(line);

WrapperDate date = entry.getDate();

Purchase purchase = entry.getPurchase();

… //обновление мэпов и листа

} catch (CsvLineException e) {

LOGGER.error("Wrong line: " + e);

}

}

...

} catch (FileNotFoundException e) {

LOGGER.fatal("couldn't open the file: " + args[0] + "\n " + e);

}

Для преобразования цикла в стримовую версию используйте следующую идиому для стрима из строк файла:

csvLine => map => Optional<WrapperEntry> => filter => Optional.isPresent() =>

map => Optional.get() => forEach => entry

словами:

замапить строку csvLine в экземпляр Optional<WrapperEntry>,

фильтровать по непустым экземплярам,

замапить в экземпляр WrapperEntry,

для каждого экземпляра entry обновить мапы и лист

try (Stream<String> csvLines = Files.lines(Paths.get(args[0]))) {

//проверьте получившийся java код, выделив 5 строк в комментах ниже:

//--- begin

csvLines

.map(WrapperEntry::from)

.filter(Optional::isPresent) // в java 9+ можно вместо этой и следующей строки

.map(Optional::get) // .flatMap(Optional::stream)

.forEach(entry -> {

//--- end

WrapperDate date = entry.getDate();

Purchase purchase = entry.getPurchase();

… //обновление мэпов и листа

});

… //пункты 5-15 с использованием стримов

} catch (FileNotFoundException e) {

LOGGER.fatal("couldn't open the file: " + args[0] + "\n" + e);

}

Таким образом, для стрима важно, чтобы строки, из которых нельзя выделить валидные сущности, мапились на пустые экземпляры Optional.

**Класс WrapperEntry**

Схематически:

public class WrapperEntry {

private final WrapperDate date;

private final Purchase purchase;

...

public static Optional<WrapperEntry> from(String csv) {

String[] values = csv.split(";", 2);

if (values.length < 2) {

log.error(...);

return Optional.empty();

}

Optional<WrapperDate> date = WrapperDate.from(values[0]);

Optional<Purchase> purchase = PurchaseFactory.getPurchase(values[1]);

проверить date и purchase;

возвратить Optional.empty() или экземпляр Optional<WrapperEntry>;

//тривиальная проверка - это вызов метода isPresent() на экземплярах date и purchase.

//решение в стиле java8 - это двойной маппинг: date.flatMap(...).map(...);

//--- begin

return date

.flatMap(wrapperDate -> purchase)

.map(purchase2 -> new WrapperEntry(wrapperDate, purchase2)));

//--- end

}

}

Таким образом, и в этот класс в альтернативных сценариях, когда scv данные не мапятся в сущности, доставляются не исключения, а пустые сущности, представленные экземплярами Optional.empty().

**Фабричный класс PurchaseFactory**

В предыдущей задаче использовалась реализация фабричного метода с помощью оператора switch. Альтернативный вариант основан на функционале перечислений.

Вспомним, что каждая константа перечисления - это константный вложенный класс. Объявите в перечислении абстрактный метод (достаточно пакетный уровень доступа), возвращающий экземпляр покупки из строки формата csv, а в каждой константе реализуйте этот метод. Тогда фабричный метод станет очень компактным.

public class PurchaseFactory {

private enum PurchaseKind {

@Override

PURCHASE {

Purchase getPurchase(String csv) {

return new Purchase(csv);

}

},

PRICE\_DISCOUNT\_PURCHASE {

...

},

PERCENT\_DISCOUNT\_PURCHASE {

...

};

abstract Purchase getPurchase(String csv);

}

public static Optional<Purchase> getPurchase(String csv) {

try {

String[] values = csv.split(";", 2);

if (values.length < 2) {

log.error(...);

return Optional.empty();

}

преобразовать values[0] в строку codeConvName;

return Optional.of(...(codeConvName).getPurchase(values[1]));

} catch (IllegalArgumentException e) {

log.error(...);

return Optional.empty();

}

}

}

Перечисление можно реализовать еще более компактно, используя ссылки на метод.

private enum PurchaseKind {

PURCHASE(Purchase::new),

PRICE\_DISCOUNT\_PURCHASE(PriceDiscountPurchase::new),

PERCENT\_DISCOUNT\_PURCHASE(PercentDiscountPurchase::new);

//------------- выделите строки между комментариями, чтобы свериться

private final Function<String, Purchase> creator;

private PurchaseKind(Function<String, Purchase> creator) {

this.creator = creator;

}

Purchase getPurchase(String csvLine) {

return creator.apply(csvLine);

}

//-------------

}

**Классы сущностей**

В этих классах также можно применить функционал Optional. Приватными статическими методами, в которых валидируются аргументы конструктора, возвращайте экземпляры Optional. Например, в классе Byn:

private static **Optional<Integer>** getValidValue(int rubs, int coins) {

if (rubs < 0 || coins < 0 || coins > 99) {

return Optional.empty();

}

return Optional.of(rubs \* 100 + coins);

}

Используя функционал Optional конструктор можно реализовать по схеме:

public Byn(int rubs, int coins) {

this(getValidValue(rubs, coins)

.**orElseThrow**(() -> new WrongArgumentsException(...))

);

}

Примените данный подход в классах всех сущностей.

Внимательно отследите условия валидности, приведенные в гайде задачи start1.

**Классы необрабатываемых пользовательских исключений**

Можно взять из задачи start1. Альтернативный вариант - это использовать единственный класс WrongArgumentsException, унаследовав его от IllegalArgumentException.

**Класс Runner**

обновить firstPurchasesMap; //2

Используйте putIfAbsent().

обновить dayPurchasesMap; //3

Используйте computeIfAbsent().

вывести каждый мэп с хедером (поясняющим комментарием); //5, 15

Реализуйте printMap() через map.forEach().

выполнить поиск покупки и вывести результат поиска; //6-8

Используйте в методе Optional.ofNullable(), а затем ifPresentOrElse().

вывести в for-each цикле для каждого entry из dayPurchasesMap

значения entry.getKey() и getTotalCost(entry.getValue()); //9

вывести getTotalCost(discountPurchases); //10

Реализуйте getTotalCost(), используя методы стрима map(), reduce() и другие.

Цикл for-each замените на dayPurchasesMap.forEach().

удалить entries; //11-14

Используйте removeIf().