Краткое summary

Тестовое покрытие измерялось при помощи JaCoCo, которое предлагает нам 2 метрики из коробки: *instruction coverage* и *branch coverage*. Все сгенерированные отчёты находятся в папке jacoco. Тестирование проводилось только на мною написанных тестах, которые находятся в соответствующих папках для lesson'ов (`TestsForSoftwareTesting`).

После запуска команды `mvn clean install` запустилось тестирование, результаты которого можно видеть ниже:

```
Results :

Failed tests:
  TestsForSoftwareTesting.testFromRoman:101 expected: <-1> but was: <1444>
  TestsForSoftwareTesting.testPlusMinus:55 Expected
java.lang.IllegalArgumentException to be thrown, but nothing was thrown.
  TestsForSoftwareTesting.testCountSubstringsGeneratedFile:65 expected: <{kek
kek=10, kek=11}> but was: <{kek=11, kek
kek=0}>

Tests run: 6, Failures: 3, Errors: 0, Skipped: 0
```

Всего было запущено 6 тестов, 3 из которых упали, потому что проверяемые функции были некорректно реализованы с точки зрения задачи.

lesson 5

В `lesson5` тестировалось две функции: `mergePhoneBooks(Map, Map)` и `containsIn(Map, Map)`. Покрытие можно видеть ниже.

MapKt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
bagPacking(Map, int)		0 %		0 %	14	14	22	22	1	1
findSumOfTwo(List, int)		0 %		0 %	17	17	16	16	1	1
averageStockPrice(List)		0 %		0 %	4	4	9	9	1	1
propagateHandshakes(Map)		0 %		0 %	8	8	15	15	1	1
canBuildFrom(List, String)		0 %		0 %	6	6	7	7	1	1
buildGrades(Map)		0 %		0 %	3	3	8	8	1	1
findCheapestStuff(Map, String)		0 %		0 %	6	6	10	10	1	1
extractRepeats(List)		0 %		0 %	4	4	11	11	1	1
filterByCountryCode(Map, String)		0 %		0 %	4	4	7	7	1	1
hasAnagrams(List)		0 %		0 %	5	5	9	9	1	1
subtractOf(Map, Map)		0 %		0 %	4	4	2	2	1	1
removeFillerWords(List, String[])		0 %		0 %	3	3	6	6	1	1
shoppingListCost(List, Map)		0 %		0 %	3	3	6	6	1	1
whoAreInBoth(List, List)		0 %		0 %	3	3	3	3	1	1
buildWordSet(List)		0 %		0 %	2	2	3	3	1	1
mergePhoneBooks(Map, Map)		100 %		100 %	0	4	0	8	0	1
containsIn(Map, Map)		100 %		100 %	0	4	0	4	0	1
Total	1 315 of 1 431	8 %	142 of 154	7 %	86	94	134	146	15	17

Так как покрытие 100% по обеим метрикам, то комментировать тут нечего.

lesson 6

В `lesson6` тестировалось две функции: `fromRoman(String)` и `plusMinus(String)`. Покрытие можно видеть ниже.

ParseKt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
dateStrToDigit(String)		0 %		0 %	9	9	27	27	1	1
dateDigitToStr(String)		0 %		0 %	7	7	25	25	1	1
flattenPhoneNumber(String)		0 %		0 %	14	14	24	24	1	1
computeDeviceCells$executeCommands(Ref.IntRef, Ref.IntRef, int, String, int, List)		0 %		0 %	13	13	20	20	1	1
bestHighJump(String)		0 %		0 %	6	6	11	11	1	1
firstDuplicateIndex(String)		0 %		0 %	7	7	13	13	1	1
computeDeviceCells$findReversedNestedLoop(String, int)		0 %		0 %	6	6	20	20	1	1
mostExpensive(String)		0 %		0 %	5	5	10	10	1	1
computeDeviceCells$findNestedLoop(String, int)		0 %		0 %	5	5	15	15	1	1
computeDeviceCells$check(String)		0 %		0 %	8	8	10	10	1	1
bestLongJump(String)		0 %		0 %	4	4	10	10	1	1
computeDeviceCells(int, String, int)		0 %		0 %	4	4	7	7	1	1
timeSecondsToStr(int)		0 %		n/a	1	1	4	4	1	1
timeStrToSeconds(String)		0 %		0 %	2	2	6	6	1	1
checkIfNotOk(String)		0 %		0 %	3	3	3	3	1	1
twoDigitStr(int)		0 %		0 %	4	4	1	1	1	1
fromRoman(String)		100 %		100 %	0	17	0	45	0	1
plusMinus(String)		100 %		90 %	3	17	0	16	0	1
safeToInt(String)		100 %		100 %	0	3	0	3	0	1
Total	1 587 of 2 214	28 %	167 of 232	28 %	101	135	206	270	16	19

В функции `plusMinus(String)` покрытие по ветвям составило 90%. Рассмотрим код программы и попробуем определить причину неполного покрытия.

```
261. fun plusMinus(expression: String): Int {
262.     if (expression.isEmpty()) throw IllegalArgumentException("Empty expression")
263.     val ok = "0123456789 "
264.     val ops = "+-"
265.     for (check in expression) if (check !in (ok + ops)) throw IllegalArgumentException("$check is not allowed in expression")
266.     if (" " !in expression) return safeToInt(expression)
267.     val commands = expression.split(" ")
268.     for (i in commands.indices) {
269.         if ((i % 2 == 0 && commands[i] in ops) || (i % 2 != 0 && commands[i] !in ops)) throw IllegalArgumentException("Bad expression format")
270.     }
271.     var res = commands[0].toInt()
272.     for (i in 1..(commands.size - 2) step 2) {
273.         val x = commands[i + 1].toInt()
274.         if (x < 0) throw IllegalArgumentException("Non-positive number in $expression")
275.         when (commands[i]) {
276.             "+" -> res += x
277.             "-" -> res -= x
278.         }
279.     }
280.     return res
281. }
```

Видно, что не полностью покрытыми являются два цикла и один `when`. Цвета ромбиков могут быть 3 типов:

> - No coverage: No branches in the line has been executed (red diamond)
> - Partial coverage: Only a part of the branches in the line have been executed (yellow diamond)
> - Full coverage: All branches in the line have been executed (green diamond)

Сначала разберёмся со вторым циклом и находящимся в нём `when`. 277 строка подсвечена жёлтым цветом, потому что у нас никогда не происходит ветвления по ветке false при сравнении с минусом. Происходит это из-за того, что по условию задачи имеется лишь 2 операции: сложение и вычитание, остальные мы не считаем корректными и выбрасываем `IllegalArgumentException`. А так как до момента захода в этот цикл обрабатываются все некорректные операции, то у нас может быть лишь 2 значения: "+" и "-". Поэтому на 277 строчке нужно было написать `else`, так как мы точно знаем, что никакие другие операции не могут присутствовать в массиве, из-за чего прохода по ветке false никогда не произойдёт:

```
when (commands[i]) {
    "+" -> res += x
    else -> res -= x
}
```

Тогда бы 272 и 277 строчки были бы окрашены в зелёный цвет.

Что касается 268 строки, то я не знаю, почему цикл подсвечен жёлтым цветом. При чём наведя курсор на жёлтый ромбик, можно видеть всплывающее окно "1 of 4 branches missed", что говорит нам о том, что какая-то одна ветка не была пройдена. В поисках объяснения в интернете я наткнулся на такой тред, где объясняется подобная ситуация. Однако в обсуждении идёт речь о форе, где всплывает окно с предупреждением "1 of 2 branches missing" с соответствующими пояснениями, а почему во всем проекте у меня во всех формах пишется про 4 ветвления, я не

могу предположить. Но даже если не брать в счёт эти 4 ветвления, то всё равно для меня остаётся загадкой подсвечивание цикла жёлтым цветом, так как есть похожий цикл на 265 строчке, где также выбрасывается исключение, однако он подсвечен зелёным цветом. Либо это баг JaCoCo, либо я чего-то не понимаю до конца.

lesson 7

В `lesson7` тестировалась лишь одна функция `countSubstrings(String, List)`, покрытие которой можно видеть ниже.

FilesKt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
markdownToHtml(String, String)		0 %		0 %	42	42	95	95	1	1
markdownToHtmlSimple(String, String)		0 %		0 %	30	30	68	68	1	1
printDivisionProcess(int, int, String)		0 %		0 %	8	8	36	36	1	1
transliterate(String, Map, String)		0 %		0 %	12	12	22	22	1	1
markdownToHtmlLists(String, String)		0 %		0 %	21	21	37	37	1	1
top20Words(String)		0 %		0 %	11	11	19	19	1	1
sibilants$correct(String)		0 %		0 %	17	17	25	25	1	1
printMultiplicationProcess(int, int, String)		0 %		0 %	3	3	29	29	1	1
centerFile(String, String)		0 %		0 %	11	11	23	23	1	1
chooseLongestChaoticWord(String, String)		0 %		0 %	7	7	12	12	1	1
sibilants(String, String)		0 %		0 %	5	5	11	11	1	1
alignFile(String, int, String)		0 %		0 %	7	7	13	13	1	1
deleteMarked(String, String)		0 %		0 %	6	6	9	9	1	1
alignFile$append(Ref.IntRef, int, BufferedWriter, String)		0 %		0 %	3	3	9	9	1	1
checkLetters(String)		0 %		0 %	3	3	6	6	1	1
isMarkdownEmpty(String)		0 %		0 %	6	6	1	1	1	1
alignFileByWidth(String, String)		0 %		n/a	1	1	1	1	1	1
pop(List)		0 %		n/a	1	1	3	3	1	1
countSubstrings(String, List)		100 %		100 %	0	7	0	15	0	1
Total	3 628 of 3 773	3 %	352 of 364	3 %	194	201	419	434	18	19

Покрытие функции составило 100%, поэтому комментировать здесь тоже нечего.