

Jacoco

В данной работе был подключен инструмент Ясосо, с помощью которого можно проанализировать код на покрытие тестами.

Инструмент подключается с помощью pom.xml файла. В конфигурации было прописано из каких файлов брать тесты, чтобы ранее написанные тесты в проекте не мешали наблюдению результатов. В частности, использовались следующие файлы:

- ChooseLongestChaoticWordTest
- PlusMinusTests
- DateStrToDigitTests
- ContainInTests
- CanBuildFromTests

При просмотре отчета об анализе мы можем наблюдать 2 значимые метрики:

- Missed Instruction Coverage
- Missed branches Coverage

Рассмотрим результаты анализа подробнее:

1. ChooseLongestChaoticWordTest

chooseLongestChaoticWord(String, String)	<div><div></div></div>	92%	<div><div></div></div>	50%
--	------------------------	-----	------------------------	-----

В данном случае покрытие кода, который был выполнен составляет 92 процента, а количество ветвлений, которое было пройдено лишь 50 процентов, если нажать на отчет, то можно увидеть в какие места были не затронуты тестами

```
1. fun chooseLongestChaoticWord(inputName: String, outputName: String) {
2.     val chosen = mutableListOf("")
3.     File(inputName).forEachLine { w ->
4.         val wLow = w.lowercase()
5.         if (wLow.all { char -> wLow.count { it == char } == 1 }) {
6.             if (w.length == chosen[0].length) chosen += w
7.             if (w.length > chosen[0].length) {
8.                 chosen.clear()
9.                 chosen += w
10.            }
11.        }
12.    }
13.    File(outputName).bufferedWriter().use { it.write(chosen.joinToString()) }
14. }
```

Соответственно здесь подчеркивается зеленым цветом, те места, которые были пройдены при тестировании, желтым цветом часть ветвлений в строке и красным если тесты сюда не заходили. В моем случае создание результирующего файла было протестировано не полностью.

2. PlusMinusTests

plusMinus(String)	<div><div></div></div>	100%	<div><div></div></div>	93%	1	9	0	11	0	1
-------------------	------------------------	------	------------------------	-----	---	---	---	----	---	---

В данном случае покрытие кода выполнено на 100%, но есть часть ветвлений куда тесты не зашли, это место будет видно на рис. ниже


```

2.  */
3.  fun plusMinus(expression: String): Int {
4.  ◆ if (expression.contains(Regex("[+-] [+-]|\d \d|^[+-]")))
5.      ◆ throw IllegalArgumentException()
6.  ◆ val stripped = expression.filter { it != ' ' }
7.      val numbers = Regex("[+-]").split(stripped)
8.  ◆ val move = stripped.filter { it in "-+" }.toList()
9.      var total = numbers[0].toInt()
0.  ◆ for (m in move.indices) {
1.          when (move[m]) {
2.          ◆ '+' -> total += numbers[m + 1].toInt()
3.          ◆ '-' -> total -= numbers[m + 1].toInt()
4.          }
5.      }
6.      return total
7.  }

```

Так как when еще имеет ветку else, куда мы никогда не заходим, то нам подсвечивает этот момент желтым. Так как мы проверяем некорректность строки в самом начале функции, а по условию имеем лишь 2 операции «+» и «-», то можно вместо '-' написать ветку элсе и тогда покрытие станет 100%.

3. DateStrToDigitTests

● dateStrToDigit(String)		100%		100%
--	---	------	---	------

Здесь покрытие ветвлений и инструкция 100%, значит тесты написаны хорошо.

4. ContainInTests и CanBuildFromTests

Здесь также аналогично тесты для обоих методов покрывают 100% кода

● canBuildFrom(List, String)		100%		100%
● containsIn(Map, Map)		100%		100%