Assignment 9

In this assignment, we performed mutation testing and analyzed the results in order to ensure the robustness of our tests and the overall code quality. First we had to fix some tests and the code related to them, so we could run the PIT library and generate the mutation report, because, as a pre-condition, all tests must pass before executing it.

Mutation Score

In the configuration, we excluded the classes related to the GUI. The report and its main packages under test are as follows:

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
34	83%	1059/1275	61%	443/725	74%	443/599

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.todotxt.todotxttouch.task	25	85%	740/875	62%	278/445	74%	278/374
com.todotxt.todotxttouch.task.sorter	2	75%	79/106	57%	43/75	81%	43/53
com.todotxt.todotxttouch.util	7	82%	240/294	60%	122/205	71%	122/172

---

com.todotxt.todotxttouch.task

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
25	85%	740/875	62%	278/445	74%	278/374

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
AndFilter.java	90%	9/10	50%	2/4	67%	2/3
ByContextFilter.java	81%	13/16	60%	6/10	86%	6/7
ByPriorityFilter.java	0%	0/12	0%	0/7	0%	0/0
ByProjectFilter.java	81%	13/16	60%	6/10	86%	6/7
ByTextFilter.java	76%	13/17	69%	9/13	90%	9/10
ContextParser.java	93%	14/15	100%	4/4	100%	4/4
FilterFactory.java	60%	9/15	0%	0/16	0%	0/11
HiddenFilter.java	0%	0/2	0%	0/2	0%	0/0
HiddenParser.java	100%	5/5	100%	3/3	100%	3/3
JdotxtTaskBagImpl.java	85%	114/134	57%	31/54	67%	31/46
LinkParser.java	82%	14/17	100%	4/4	100%	4/4
LocalFileTaskRepository.java	55%	33/60	14%	4/29	24%	4/17
MailAddressParser.java	92%	12/13	100%	4/4	100%	4/4
OrFilter.java	0%	0/12	0%	0/7	0%	0/0
PhoneNumberParser.java	100%	5/5	100%	1/1	100%	1/1
Priority.java	98%	60/61	91%	21/23	95%	21/22
PriorityTextSplitter.java	93%	13/14	80%	4/5	100%	4/4
ProjectParser.java	93%	14/15	100%	4/4	100%	4/4
RecParser.java	100%	12/12	100%	3/3	100%	3/3
Task.java	96%	209/217	74%	121/164	76%	121/159
TaskBagFactory.java	0%	0/3	0%	0/1	0%	0/0
TaskBagImpl.java	84%	113/135	62%	34/55	76%	34/45
TextSplitter.java	98%	42/43	90%	9/10	100%	9/9
ThresholdDateFilter.java	80%	4/5	20%	1/5	25%	1/4
ThresholdDateParser.java	90%	19/21	100%	7/7	100%	7/7

---

com.todotxt.todotxttouch.task.sorter

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	75%	79/106	57%	43/75	81%	43/53

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
GenericSorter.java	88%	7/8	40%	2/5	50%	2/4
Sorters.java	73%	72/98	59%	41/70	84%	41/49

---

com.todotxt.todotxttouch.util

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
7	82%	240/294	60%	122/205	71%	122/172

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CursorPositionCalculator.java	89%	8/9	80%	8/10	80%	8/10
Path.java	92%	12/13	100%	11/11	100%	11/11
RelativeDate.java	52%	13/25	4%	1/28	8%	1/13
Strings.java	96%	45/47	73%	30/41	75%	30/40
TaskIo.java	82%	51/62	67%	26/39	74%	26/35
Tree.java	85%	33/39	74%	14/19	82%	14/17
Util.java	79%	78/99	56%	32/57	70%	32/46

Mutants Analysis

We found many instances where the conditional boundary mutant survived, this mutant changes the conditional operator to a boundary counterpart:

```
53 2        pos = pos < 0 ? 0 : pos;
54 3        return pos > newValue.length() ? newValue.length() : pos;
```

```
53  1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
    1. changed conditional boundary → SURVIVED
```

In some cases, the mutants survived to the replacement of the return value, which means that part of our tests could be testing some internal logic but were not checking the final return value, or did not explore all the return value possibilities.

```
1. replaced boolean return with false for com/todotxt/todotxttouch/task/Task::isHidden → SURVIVED
```

Other major part of mutants that survived are from negated conditionals. It replaces the conditionals for inverse ones, for instance, `!=` becomes `==`.

```
112 1           isFromThreshold = !(parsedRec[0].isEmpty());
```

```
112  1. negated conditional → SURVIVED
```

Also, a considerable amount of coverage was lost due to unreachable statements. That's the case of the `RelativeDate` class, where most of the code has no coverage, thus a high survivability compared with other parts of the source code.

Equivalent mutants are mutants that change some part of the code, but it still behaves as expected. They should be manually identified and ignored if possible. This conditional boundary mutant found in the `CursorPositionCalculator` is equivalent and behaves as the original code. Replacing `<` for `<=` will still assign the integer 0:

```
53 2          pos = pos < 0 ? 0 : pos;
```

The same equivalent style happens in the `FilterFactory` class, where independently of the operator being `>` or `>=` it maintains the same behaviour if there are contexts or not, because if the `ByContextFilter` receives an empty list it will always evaluate to true when being applied.

```
45 2        if (contexts.size() > 0) {
46 1            filter.addFilter(new ByContextFilter(contexts));
```

Increasing mutation score and tests developed

In the end, we obtained a total of 66% of mutation coverage. Although it is not very high, there are some reasons: Firstly, there are a significant amount of NO_COVERAGE entries, and simply covering those lines would not be easy:

```
51  1. replaced return value with "" for com/todotxt/todotxttouch/util/RelativeDate::getRelativeDate → SURVIVED
54  1. changed conditional boundary → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
56  1. Replaced long division with multiplication → NO_COVERAGE
57  1. replaced return value with "" for com/todotxt/todotxttouch/util/RelativeDate::getRelativeDate → NO_COVERAGE
60  1. changed conditional boundary → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
62  1. replaced return value with "" for com/todotxt/todotxttouch/util/RelativeDate::getRelativeDate → NO_COVERAGE
65  1. changed conditional boundary → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
67  1. Replaced long division with multiplication → NO_COVERAGE
68  1. replaced return value with "" for com/todotxt/todotxttouch/util/RelativeDate::getRelativeDate → NO_COVERAGE
71  1. changed conditional boundary → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
73  1. replaced return value with "" for com/todotxt/todotxttouch/util/RelativeDate::getRelativeDate → NO_COVERAGE
77  1. replaced return value with "" for com/todotxt/todotxttouch/util/RelativeDate::getRelativeDate → NO_COVERAGE
89  1. removed call to java/util/Calendar::setTime → SURVIVED
```

In the above example, we can see that the RelativeDate Class is practically not covered. However, developing tests for these lines would require a lot of mocking, and since this class's usage is only tied with our developed tests, these tests would serve no real purpose.

```
71  1. replaced int return with 0 for com/todotxt/todotxttouch/task/sorter/Sorters$4$1::compare → NO_COVERAGE
79  1. replaced return value with null for com/todotxt/todotxttouch/task/sorter/Sorters$5::get → NO_COVERAGE
82  1. replaced int return with 0 for com/todotxt/todotxttouch/task/sorter/Sorters$5$1::compare → NO_COVERAGE
90  1. replaced return value with null for com/todotxt/todotxttouch/task/sorter/Sorters$6::get → NO_COVERAGE
96  1. Replaced integer multiplication with division → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
    3. replaced int return with 0 for com/todotxt/todotxttouch/task/sorter/Sorters$6$1::compare → NO_COVERAGE
104 1. replaced return value with null for com/todotxt/todotxttouch/task/sorter/Sorters$7::get → NO_COVERAGE
110 1. Replaced integer multiplication with division → NO_COVERAGE
    2. negated conditional → NO_COVERAGE
    3. replaced int return with 0 for com/todotxt/todotxttouch/task/sorter/Sorters$7$1::compare → NO_COVERAGE
118 1. replaced return value with null for com/todotxt/todotxttouch/task/sorter/Sorters$8::get → NO_COVERAGE
121 1. replaced int return with 0 for com/todotxt/todotxttouch/task/sorter/Sorters$8$1::compare → NO_COVERAGE
```

Another obstacle that we faced, as explained in a previous assignment, is that there are a lot of private methods with and without coverage, and we have no easy way to develop a test for these methods, since we would need to either change the code or try our luck with reflection-based tests or try to test functions that indirectly call these methods.

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
34	86%	1095/1275	66%	477/725	76%	477/624

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.todotxt.todotxttouch.task	25	88%	771/875	70%	311/445	78%	311/398
com.todotxt.todotxttouch.task.sorter	2	75%	79/106	57%	43/75	81%	43/53
com.todotxt.todotxttouch.util	7	83%	245/294	60%	123/205	71%	123/173

Overall, the tests created were to cover parts of the code that were reachable but were missing tests. We have also added tests to handle some mutants such as the conditional boundary mutants and thus increase our Mutation Coverage by only 5% as seen in the data above.