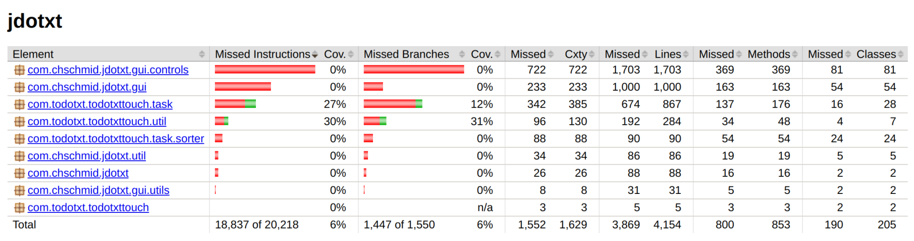
Assignment 5

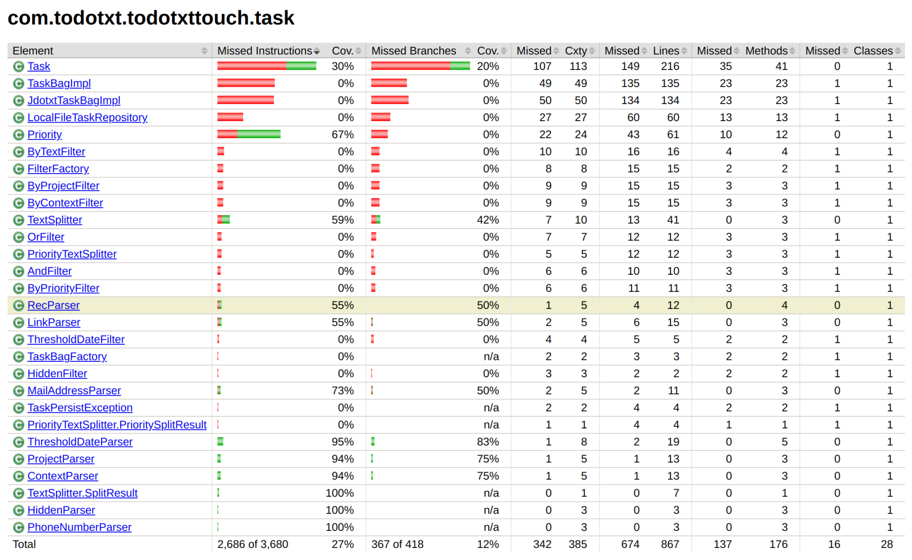
Line and Branch Coverage

The GUI related modules are entirely not exercised by any tests, but some GUI features were tested in the previous assignment. The Task and Util modules have the highest coverage, but still cannot reach more than half of the code and branches in it:



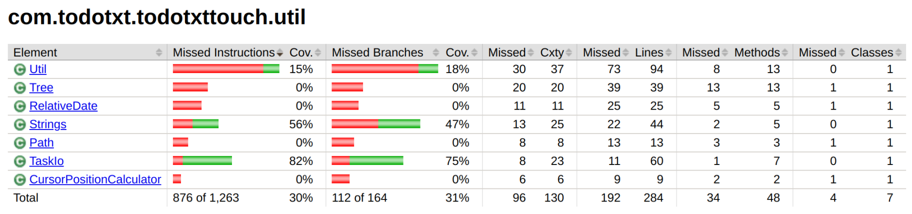
jdotxt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.chschmid.jdotxt.gui.controls		0%		0%	722	722	1,703	1,703	369	369	81	81
com.chschmid.jdotxt.gui		0%		0%	233	233	1,000	1,000	163	163	54	54
com.todotxt.todotxttouch.task		27%		12%	342	385	674	867	137	176	16	28
com.todotxt.todotxttouch.util		30%		31%	96	130	192	284	34	48	4	7
com.todotxt.todotxttouch.task.sorter		0%		0%	88	88	90	90	54	54	24	24
com.chschmid.jdotxt.util		0%		0%	34	34	86	86	19	19	5	5
com.chschmid.jdotxt		0%		0%	26	26	88	88	16	16	2	2
com.chschmid.jdotxt.gui.utils		0%		0%	8	8	31	31	5	5	2	2
com.todotxt.todotxttouch		0%		n/a	3	3	5	5	3	3	2	2
Total	18,837 of 20,218	6%	1,447 of 1,550	6%	1,552	1,629	3,869	4,154	800	853	190	205



com.todotxt.todotxttouch.task

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Task		30%		20%	107	113	149	216	35	41	0	1
TaskBagImpl		0%		0%	49	49	135	135	23	23	1	1
JdotxtTaskBagImpl		0%		0%	50	50	134	134	23	23	1	1
LocalFileTaskRepository		0%		0%	27	27	60	60	13	13	1	1
Priority		67%		0%	22	24	43	61	10	12	0	1
ByTextFilter		0%		0%	10	10	16	16	4	4	1	1
FilterFactory		0%		0%	8	8	15	15	2	2	1	1
ByProjectFilter		0%		0%	9	9	15	15	3	3	1	1
ByContextFilter		0%		0%	9	9	15	15	3	3	1	1
TextSplitter		59%		42%	7	10	13	41	0	3	0	1
OrFilter		0%		0%	7	7	12	12	3	3	1	1
PriorityTextSplitter		0%		0%	5	5	12	12	3	3	1	1
AndFilter		0%		0%	6	6	10	10	3	3	1	1
ByPriorityFilter		0%		0%	6	6	11	11	3	3	1	1
RecParser		55%		50%	1	5	4	12	0	4	0	1
LinkParser		55%		50%	2	5	6	15	0	3	0	1
ThresholdDateFilter		0%		0%	4	4	5	5	2	2	1	1
TaskBagFactory		0%		n/a	2	2	3	3	2	2	1	1
HiddenFilter		0%		0%	3	3	2	2	2	2	1	1
MailAddressParser		73%		50%	2	5	2	11	0	3	0	1
TaskPersistException		0%		n/a	2	2	4	4	2	2	1	1
PriorityTextSplitter.PrioritySplitResult		0%		n/a	1	1	4	4	1	1	1	1
ThresholdDateParser		95%		83%	1	8	2	19	0	5	0	1
ProjectParser		94%		75%	1	5	1	13	0	3	0	1
ContextParser		94%		75%	1	5	1	13	0	3	0	1
TextSplitter.SplitResult		100%		n/a	0	1	0	7	0	1	0	1
HiddenParser		100%		n/a	0	3	0	3	0	3	0	1
PhoneNumberParser		100%		n/a	0	3	0	3	0	3	0	1
Total	2,686 of 3,680	27%	367 of 418	12%	342	385	674	867	137	176	16	28



com.todotxt.todotxttouch.util

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Util		15%		18%	30	37	73	94	8	13	0	1
Tree		0%		0%	20	20	39	39	13	13	1	1
RelativeDate		0%		0%	11	11	25	25	5	5	1	1
Strings		56%		47%	13	25	22	44	2	5	0	1
Path		0%		0%	8	8	13	13	3	3	1	1
TaskIo		82%		75%	8	23	11	60	1	7	0	1
CursorPositionCalculator		0%		0%	6	6	9	9	2	2	1	1
Total	876 of 1,263	30%	112 of 164	31%	96	130	192	284	34	48	4	7

Additional Tests

The generated were focused on the two main elements of the project, that are not related to the GUI: the `task` and `util` module.

To make it possible to test more parts of the code and more efficiently, we used some distinct features of the JUnit package and also made use of Mockito, as some methods depended on other static values and methods of the GUI module, which made it difficult to test without mocks.

Organize In Distinct Test Suites To separate different methods and its tests, we created a suite with the **@Suite** annotation and defined classes to contain the tests for each method.

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
        StringsTest.InsertPaddedIfNeededTest.class,
        StringsTest.InsertPaddedTest.class,
        StringsTest.IsBlankTest.class
})
public class StringsTest {

    public static class InsertPaddedIfNeededTest {
        ...
    }

    public static class InsertPaddedTest {
        ...
    }

    public static class IsBlankTest {
        ...
    }
}
```

We used it in some occasions to help with the organization and clarity of the tests.

Expected Exceptions We have also added tests that test for an expected Exception thrown by the in test method. In order to check for an exception in our program, we used the **expected** parameter of the **@Test** annotation:

```
    @Test(expected = NullPointerException.class)
    public void testPrioritySorterWithNullTask() {
        Sorter<Task> gs = Sorters.PRIORITY.get(true);
        Task t1 = null;
        Task t2 = new Task(1,"Test task!");
        assertEquals(gs.compare(t1,t2),0);
    }
```

Parameterized To easily execute multiple test cases, we run these repetitive tests with the support of the **@Parameterized** annotation family, which conveys an interface to define a collection of inputs and its expected values, and then test it with our test methods:

```
@RunWith(Parameterized.class)
    public static class EqualityTest {

        enum Type {EQUAL, NOT_EQUAL};

        @Parameterized.Parameters
        public static Collection input() {
            return Arrays.asList(new Object[][] {
                    {
                            Type.EQUAL,
                            new Task(0, "x (A) Task 1 +project @context"),
                            new Task(0, "x (A) Task 1 +project @context")
                    },
                    {
                            Type.NOT_EQUAL,
                            new Task(0, "(A) Task 1"),
                            new Task(0, "(A) Task 1 +project")
                    },
                    // ...
            });
        }

        @Parameterized.Parameter()
        public Type type;

        @Parameterized.Parameter(1)
        public Task taskInput;

        @Parameterized.Parameter(2)
        public Task taskExpected;

        @Test
        public void equal() {
            Assume.assumeTrue(type==Type.EQUAL);
            assertEquals(taskExpected, taskInput);
            assertEquals(taskExpected.hashCode(), taskInput.hashCode());
        }

        @Test
        public void notEqual() {
            Assume.assumeTrue(type==Type.NOT_EQUAL);
            assertFalse(taskInput.equals(taskExpected));
            if(taskExpected != null)
```

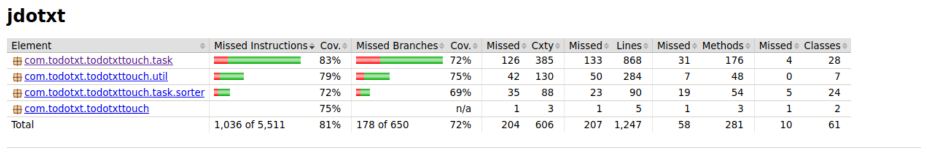
```
                assertNotEquals(taskExpected.hashCode(), taskInput.hashCode());
        }
    }
```

Mockito Mockito is a testing framework that enable us to create mocks, that can be objects with part of its behaviour modified in order to make it easier to test with other parts of the code that depends on it.

For instance, when creating a task with dates, internally the module would call the `RelativeDate.getRelativeDate` static method, which in turn calls a method from a variable of the GUI module, this variable if not defined, would create an exception and the program would crash. We agreed that this behaviour was a design flaw, and that test should indeed fail, however, in order to increase the coverage and because it was needed in multiple tests, we used the mocks as a workaround for this problem.

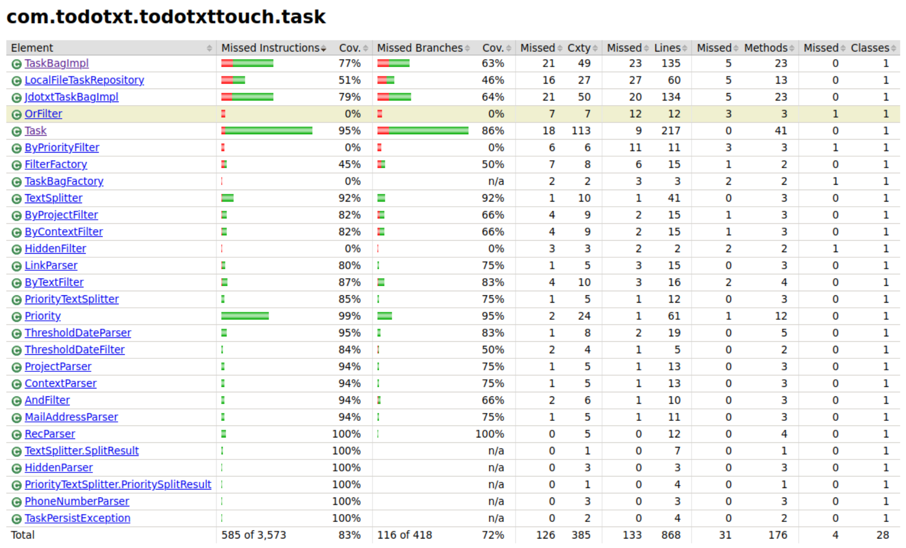
```
    @Test
    public void testArchive() {
        try (MockedStatic<RelativeDate> classMock = mockStatic(RelativeDate.class)) {
            classMock.when(
                () -> RelativeDate.getRelativeDate(
                    any(Date.class)
                )
            ).thenReturn("");
            // ...
        }
    }
```

Line and Branch Coverage After Additional tests After the additional tests, we were able to significantly increase the tests coverage (for this analysis, we have excluded the GUI related module):



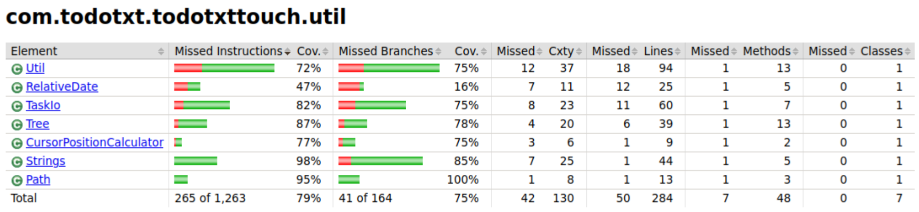
jdotxt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.todotxt.todotxttouch.task	🟥🟩	83%	🟥🟩	72%	126	385	133	868	31	176	4	28
com.todotxt.todotxttouch.util	🟥🟩	79%	🟥🟩	75%	42	130	50	284	7	48	0	7
com.todotxt.todotxttouch.task.sorter	🟥🟩	72%	🟥🟩	69%	35	88	23	90	19	54	5	24
com.todotxt.todotxttouch		75%		n/a	1	3	1	5	1	3	1	2
Total	1,036 of 5,511	81%	178 of 650	72%	204	606	207	1,247	58	281	10	61



com.todotxt.todotxttouch.task

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TaskBagImpl		77%		63%	21	49	23	135	5	23	0	1
LocalFileTaskRepository		51%		46%	16	27	27	60	5	13	0	1
JdotxtTaskBagImpl		79%		64%	21	50	20	134	5	23	0	1
OrFilter		0%		0%	7	7	12	12	3	3	1	1
Task		95%		86%	18	113	9	217	0	41	0	1
ByPriorityFilter		0%		0%	6	6	11	11	3	3	1	1
FilterFactory		45%		50%	7	8	6	15	1	2	0	1
TaskBagFactory		0%		n/a	2	2	3	3	2	2	1	1
TextSplitter		92%		92%	1	10	1	41	0	3	0	1
ByProjectFilter		82%		66%	4	9	2	15	1	3	0	1
ByContextFilter		82%		66%	4	9	2	15	1	3	0	1
HiddenFilter		0%		0%	3	3	2	2	2	2	1	1
LinkParser		80%		75%	1	5	3	15	0	3	0	1
ByTextFilter		87%		83%	4	10	3	16	2	4	0	1
PriorityTextSplitter		85%		75%	1	5	1	12	0	3	0	1
Priority		99%		95%	2	24	1	61	1	12	0	1
ThresholdDateParser		95%		83%	1	8	2	19	0	5	0	1
ThresholdDateFilter		84%		50%	2	4	1	5	0	2	0	1
ProjectParser		94%		75%	1	5	1	13	0	3	0	1
ContextParser		94%		75%	1	5	1	13	0	3	0	1
AndFilter		94%		66%	2	6	1	10	0	3	0	1
MailAddressParser		94%		75%	1	5	1	11	0	3	0	1
RecParser		100%		100%	0	5	0	12	0	4	0	1
TextSplitter.SplitResult		100%		n/a	0	1	0	7	0	1	0	1
HiddenParser		100%		n/a	0	3	0	3	0	3	0	1
PriorityTextSplitter.PrioritySplitResult		100%		n/a	0	1	0	4	0	1	0	1
PhoneNumberParser		100%		n/a	0	3	0	3	0	3	0	1
TaskPersistException		100%		n/a	0	2	0	4	0	2	0	1
Total	585 of 3,573	83%	116 of 418	72%	126	385	133	868	31	176	4	28



com.todotxt.todotxttouch.util

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Util		72%		75%	12	37	18	94	1	13	0	1
RelativeDate		47%		16%	7	11	12	25	1	5	0	1
TaskIo		82%		75%	8	23	11	60	1	7	0	1
Tree		87%		78%	4	20	6	39	1	13	0	1
CursorPositionCalculator		77%		75%	3	6	1	9	1	2	0	1
Strings		98%		85%	7	25	1	44	1	5	0	1
Path		95%		100%	1	8	1	13	1	3	0	1
Total	265 of 1,263	79%	41 of 164	75%	42	130	50	284	7	48	0	7

Nevertheless, we were not able to perform more than 90% of both code and branch coverages. Multiple tests that we have created, while increased the code coverage, did not actually create significant value for the program verification. For instance, we had a method `equals` that was responsible for checking the equality of the **Task** class, and to be able to check every branch of this method, we had to use Java Reflections to modify the values of attributes of this class at runtime, thus increasing the coverage. However, these lines and branches were very unlikely to happen in the normal flow of the program, as the way the data is passed between the objects, some null attribute values were not possible to obtain without changing the internal elements and behaviour of a class. In addition to that, there were multiple methods that relied on static variables and methods from other classes, and extensively testing these methods would require an excessive usage of mocks.