Nathaniel Anderson
12/9/22
422 – Deliverable 3

Deliverable 3 Report

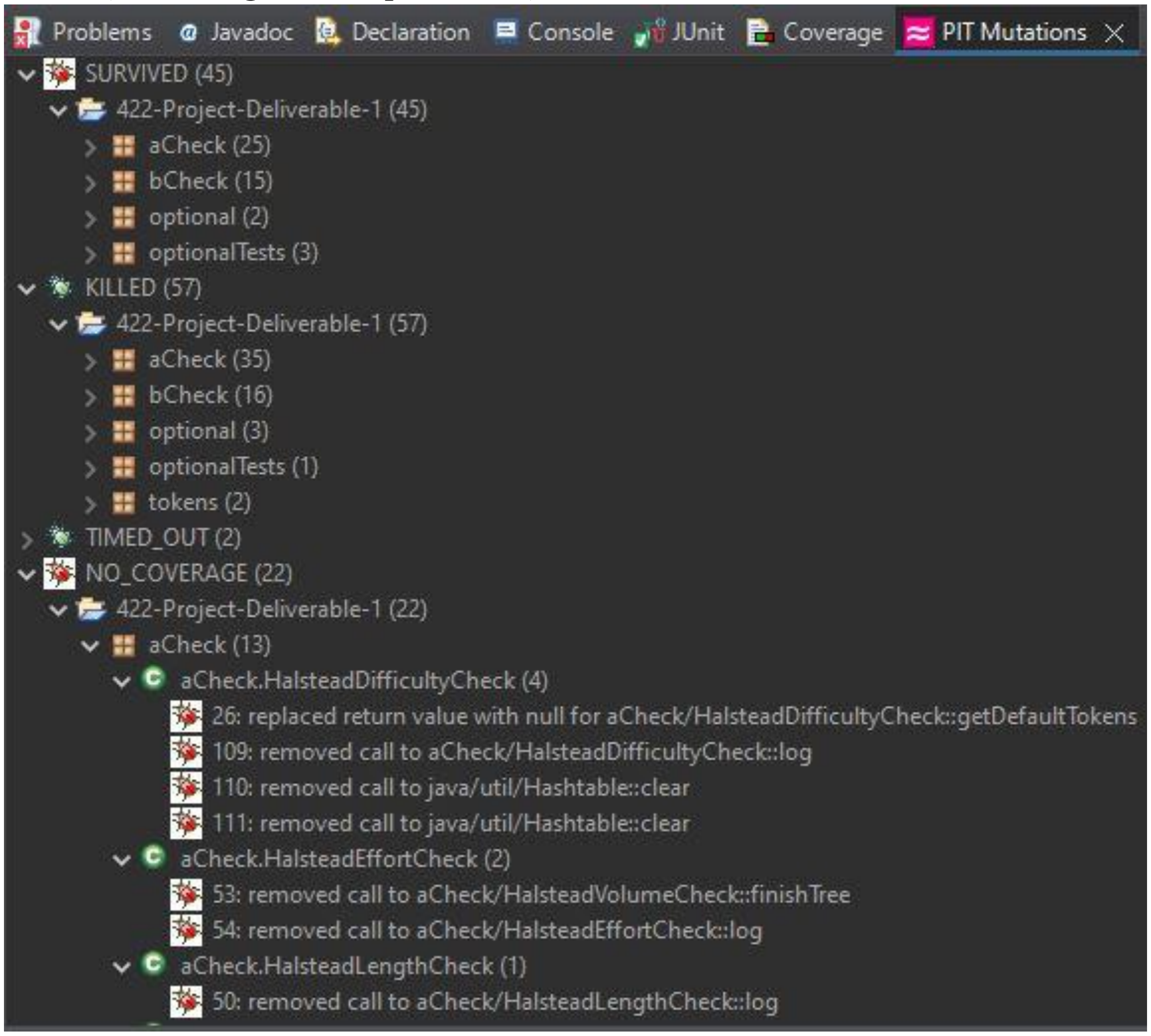
(Succinct Explanation of Tests/Checks in ReadMe)

Evaluation

In my last deliverable I checked the coverage with Junit testing 5. After fixing/omitting a couple of Tests I had an overall coverage of 91.3% this time around (Details on this later, after test improvements and implementations).

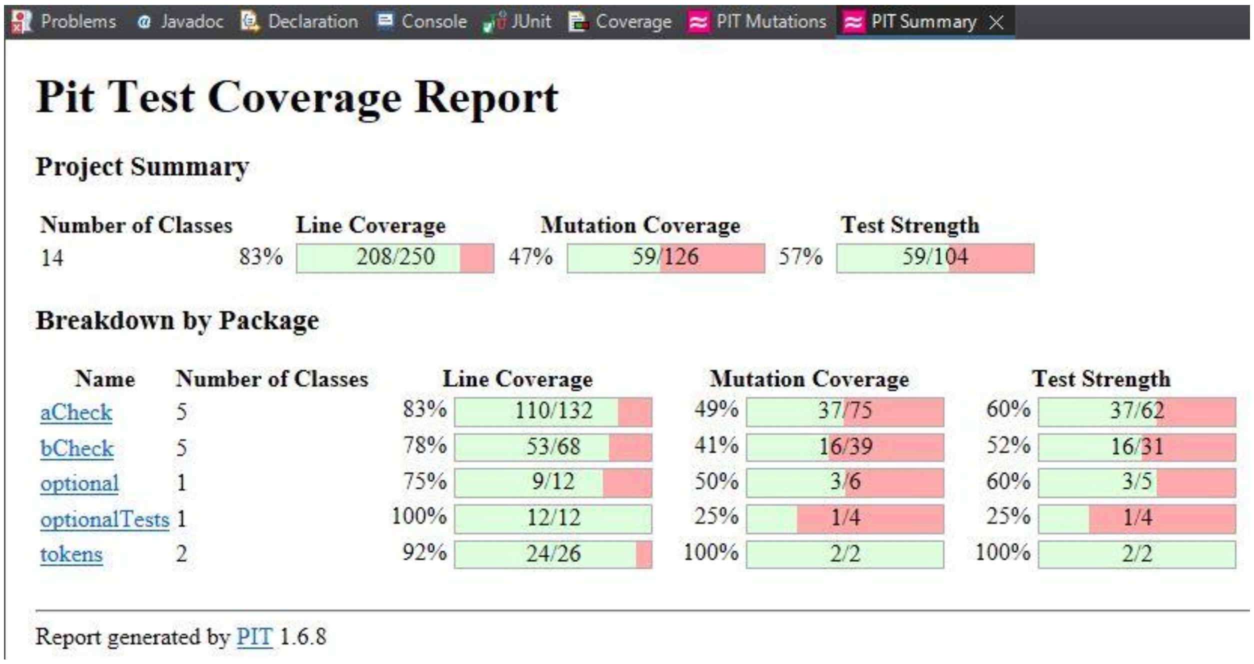
For evaluation I also used PIT Mutation tests and took screenshots of the coverage and bugs that survived, were killed, and were not covered. This Evaluation was done before and after the black box tests:

Before (BB Testing/Code Improvement):



```
SURVIVED (45)
  422-Project-Deliverable-1 (45)
    aCheck (25)
    bCheck (15)
    optional (2)
    optionalTests (3)
KILLED (57)
  422-Project-Deliverable-1 (57)
    aCheck (35)
    bCheck (16)
    optional (3)
    optionalTests (1)
    tokens (2)
TIMED_OUT (2)
NO_COVERAGE (22)
  422-Project-Deliverable-1 (22)
    aCheck (13)
      aCheck.HalsteadDifficultyCheck (4)
        26: replaced return value with null for aCheck/HalsteadDifficultyCheck::getDefaultTokens
        109: removed call to aCheck/HalsteadDifficultyCheck::log
        110: removed call to java/util/Hashtable::clear
        111: removed call to java/util/Hashtable::clear
      aCheck.HalsteadEffortCheck (2)
        53: removed call to aCheck/HalsteadVolumeCheck::finishTree
        54: removed call to aCheck/HalsteadEffortCheck::log
      aCheck.HalsteadLengthCheck (1)
        50: removed call to aCheck/HalsteadLengthCheck::log
```

Figure 1 – shows survived bugs (45), killed bugs (57), bugs not covered (22)- all done before Black Box Testing.



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
14	83%	208/250	47%	59/126	57%	59/104

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
aCheck	5	83%	110/132	49%	37/75	60%	37/62
bCheck	5	78%	53/68	41%	16/39	52%	16/31
optional	1	75%	9/12	50%	3/6	60%	3/5
optionalTests	1	100%	12/12	25%	1/4	25%	1/4
tokens	2	92%	24/26	100%	2/2	100%	2/2

Report generated by PIT 1.6.8

Figure 2 – shows the coverage report of the Mutation tests.

As we can observe in the two figures above, we have OK code coverage, with over 83% line coverage and over half is mutation covered. We will be revisting this evaluation technique after making changes to improve the code.
Next, I made a couple fault models and list the potential black box tests to be written for the aChecks and bChecks.

Fault Models/BlackBox Test Assumptions:

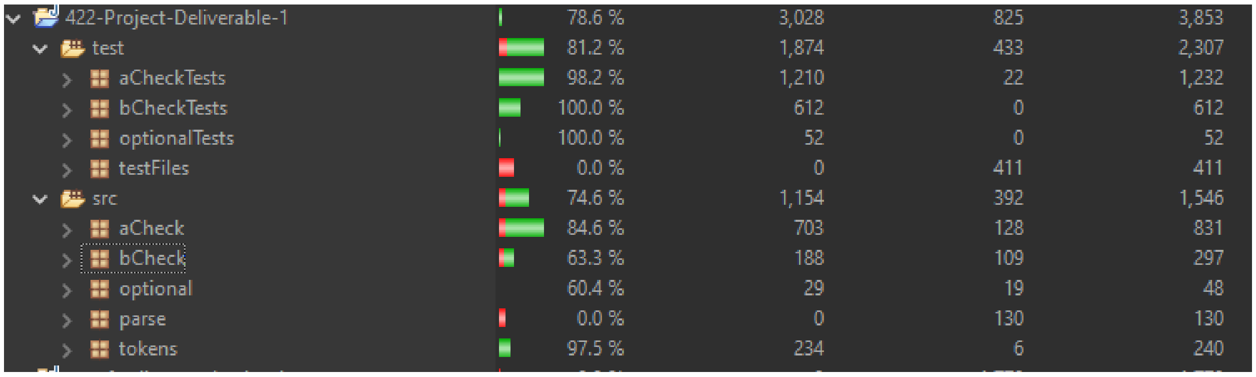
First, a general model for each category of Checks. Then based on these general fault models we can write more specific black box.

A. Halstead Checks (aChecks)
- Large number of operands/operands
- Repeating operand/operator names
- Length problems
- Wrong calculation in effort

B. General Metrics (bChecks)
- Null count (i.e. no operators)
- Single count (i.e. one comment)
- Large count (i.e. many operators or comments)
- Nested elements (comment, loops)
- Multiple lines for one count (loops, comments)

Now that I have a model of potential situations causing faults, I can now write tests to cover these potential problems, using black box methodology.

After (BB Testing/Code Improvement):



422-Project-Deliverable-1	78.6 %	3,028	825	3,853
test	81.2 %	1,874	433	2,307
aCheckTests	98.2 %	1,210	22	1,232
bCheckTests	100.0 %	612	0	612
optionalTests	100.0 %	52	0	52
testFiles	0.0 %	0	411	411
src	74.6 %	1,154	392	1,546
aCheck	84.6 %	703	128	831
bCheck	63.3 %	188	109	297
optional	60.4 %	29	19	48
parse	0.0 %	0	130	130
tokens	97.5 %	234	6	240

Figure 3 - New coverage after adding all the testfiles and fixing one code issue with getDefaultTokens() (without bb tests)

Running out of time, I was unable to run Evaluations after writing the code for Black Box testing because of one single error. The puppy crawl getLines() was coming back as null when I tried to parse any of my test files using a modified Tree Walker I made. After about an hour of research and troubleshooting, I decided to move on. This is very discouraging as I believe I have the Black Box testing structure down, and I hope I can receive some credit for the way I went about making/structuring the tests.

In theory, the black box tests created would have killed more mutations and increased overall coverage.

I also tried to improve my code coverage last minute by trying to fix the finish tree testing, but still was unable to figure it out. (Example of different solution in ExpressionCheckTest.Java)

I ran the Pit Mutation tests to evaluate my code, and this was done before any major test/check improvements. Because of this I had to omit two tests for the PIT to run, getOperatorsTest() and getCommentsCountTest(). These were passing the Junit tests no problem, but for some reason PIT didn't like them and because PIT needs a green suite, it wouldn't go thru with the Mutation Testing.

Class Testing:

Class testing is more involved than Basic Unit testing. Class testing tests the status and states of variables/classes. For me to incorporate this, I would have to re-evaluate existing tests. This means incorporating a status tracking method with a new "getter" of sorts to evaluate the existing tests as the program executes. I would also have to implement a series of new tests to evaluated conditions of the class. I may have to draw out state trees first and state charts. Doing this as a preliminary step would help with the overall development of these types of tests. I don't have a lot of class dependency so inter-class testing won't be as prominent. I would focus more on intra-class testing. Also, figuring out a way to test the system as a whole, especially when this is being used to black box test, might increase coverage/mitigate faults and/or bugs.