Deliverable 3
Tony Barrera

1) ReadME.md has been included
2) Evaluate your own tests in two steps:
    a. Measuring statement and branch coverage is in the deliverable 2 folder. All classes were able to achieve 100% line and branch coverage.
    b. Running mutation Tests. Using pitclipse: From deliverable 2, line an branch coverage was all 100%, after running mutation tests with pitclipse these were my results.

Class Name	%	Line Coverage	%	Mutation Coverage	%	Test Strength
ACheck – HalsteadLength	100%	16/16	88%	7/8	88%	7/8
Halstead Vocabulary	100%	15/15	86%	6/7	86%	6/7
Halstead Volume	100%	22/22	85%	11/13	85%	11/13
Halstead Effort	100%	41/41	95%	20/21	95%	20/21
Halstead Difficulty	100%	37/37	62%	8/13	62%	8/13
BCheck - NumberOfComments	100%	16/16	88%	7/8	88%	7/8
CommentLineCount	100%	17/17	88%	7/8	88%	7/8
LoopStatementCount	100%	19/19	89%	8/9	89%	8/9
OperandCount	100%	23/23	89%	8/9	89%	8/9
OperatorCount	100%	17/17	88%	7/8	88%	7/8
ExpressionCount	100%	14/14	88%	7/8	88%	7/8
TypecastCount	100%	16/16	89%	8/9	89%	8/9

It seems like the majority of yellow tests where there's only one mutation that did not go well was "remove call to the AbstractCheck::Log(…)", which is the one I was having trouble with to get 100% line coverage in deliverable 2. The reason it is failing because mutation test is expecting the tests to fail when log() is removed, so how do I go about that? I put my thinking cap on and started brain storming, but there was not much I could think of because log does nothing that we care about (at least not in terms of the standalone test). Not sure what I am missing here. It seems like something needs to change the value IF we DON'T execute the log() method.

One of the mutations that survived was when pitclipse replaces / with *, and this was because my test case was using the number '1', which gave us the same result. I needed to change the test to use different values in order to get different results when using * as opposed to /.

The class that needed the most work was Halstead Difficulty with a mutation score of 62%, here are all the changes I made to increase the mutation coverage:

- Difficulty implementation was implemented wrong and I did not update it once I updated it in my "Effort" calculation when I initially caught it for Del2.

This brings the HalsteadDifficultyCheck Mutation coverage to **12/13 as opposed to 8/13.**

Side Note: Just realized I could run all my tests at once and didn't need to make that needlessly long chart???????????

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
ACheck.java	100%	16/16	88%	7/8	88%	7/8
AntiHungarianCheck.java	0%	0/17	0%	0/13	0%	0/0
BCheck.java	100%	16/16	88%	7/8	88%	7/8
CastCountCheck.java	100%	16/16	89%	8/9	89%	8/9
CommentLineCountCheck.java	100%	17/17	88%	7/8	88%	7/8
ExpressionCountCheck.java	100%	14/14	88%	7/8	88%	7/8
HalsteadArrayMaster.java	95%	105/110	67%	2/3	100%	2/2
HalsteadDifficultyCheck.java	100%	39/39	92%	12/13	92%	12/13
HalsteadEffortCheck.java	100%	41/41	95%	20/21	95%	20/21
HalsteadVocabularyCheck.java	100%	15/15	86%	6/7	86%	6/7
HalsteadVolumeCheck.java	100%	22/22	92%	12/13	92%	12/13
LoopStatementCountCheck.java	100%	19/19	89%	8/9	89%	8/9
MethodLimitCheck.java	0%	0/12	0%	0/6	0%	0/0
OperandCountCheck.java	100%	23/23	89%	8/9	89%	8/9
OperatorCountCheck.java	100%	17/17	88%	7/8	88%	7/8
SemicolonCountCheck.java	0%	0/12	0%	0/6	0%	0/0

Black Box Tests & Fault Models:

Number of Comments: The more likely metric to miss in this test is confusing multiline comments and single line comments where the body of comments gets counted by accident.

1) Three Comments
2) One Comment

Number of lines of comments: like above, the confusion could lie in the distinction of lines of comments, so the tokens need to be properly declared to get the correct results:

1) Three Comments file which contains 5 lines of comments
2) One comment file which contains 3 lines of code

Loop Count: this one is pretty straight forward as we only need to check for the major loops, for loop, while loop, and do while.

1) No loops
2) Three different loops, for loop, while loop, and do while.

OperandCount:

1) Zero operand
2) Two operand

OperatorCount:

1) Zero operator
2) One operator

ExpressionCount:

1) Zero expression
2) One Expression

TypecastCount:

1) Zero typecast

2) One typecast