Deliverable Three

The sample plug library delivered contains the following files relevant to blackbox testing:

BlackBoxTestEngine.java, which runs and tests the expected output of all the tests below

Tested on comment number and comment lines checkers:

CommentBothBlackTest.java - has both kinds of comments
CommentFaultBlackTest.java - has fault comments
CommentMultiLineBlackTest.java - has just multiline comments
CommentSingleBlackTest.java - has just single line comments

Tested on ALL checkers:
EmptyBlackTest.java - an empty java file - used on ALL checkers to make sure they return 0 on an empty java file

Tested on Expression checker:
ExpressionBlackTest.java -
ExpressionFaultBlackTest.java -

Tested on loop checker:
LoopDoBlackTest.java
LoopFaultBlackTest.java
LoopForBlackTest.java
LoopNestedBlackTest.java
LoopWhileBlackTest.java

Tested on all 5 Halstead checkers:
HalFaultBlackTest.java

Tested on operator checker, operand checker, and all Halstead checkers (since all those have to do with operands and operators):

OpFaultBlackTest.java
OpManyBlackTest.java
OpSingleBlackTest.java

As you may notice, we use the same java files to check different checks, since many of them have the same things they are looking for (some all check for comments, or operands, operators, etc)

Each subject (op, Hal, loop, etc) has a FAULT black test. This FAULT black test is used to test possible faults that may exist in the check style checker code we are testing. These tests usually contain multiple faults, usually just making sure that certain operands/operators aren't being counted as distinct from one another, etc. More detail will be given on this below.

As a rule, most check groups have a check for a SINGLE of what to look for (with some having multiple types), MANY of what to look for, MULTI of what to look for (different kinds of loops, different kinds of ops). In some cases, like the loop check, we test for loop, do loop, and while loop separately, and then combined in NESTED.

Procedure (similar in all test cases):

- Sets up the necessary configurations and contextualizes checks.
- Initializes local variables in the check.
- Visits each token in the tree with a single operator.
- Completes the tree and displays intended logs to the user.
- Verifies the result for the Halstead Length Check, expecting a certain number.

If we took a class based approach (rather than blackbox/unit based):

Since our check style checks interact with the check style parent class, we would need to make test methods that we inherit from that, even if we don't explicitly write them out in our own java code. We would still want to evaluate our ability to handle different kinds of code (empty, faulty, etc). We would move beyond the individual method validations to assess the overall functionality, which would likely include testing all classes which our class uses. This would be much more exhaustive. It would be more about how our class acts as a result of inheriting from it's parent class, and how it interacts with the DetailAST objects, and what other classes interact with that DetailAST class we are dealing with, which could prove to have an important effect. It might even branch into testing the parsing classes which would turn the java files into DetailAST's, since that is also a vital part of a real world user being able to test their java file.

Halstead Length Tests

1. **halsteadLengthEmptyTest:**
    - **Purpose:** Checks Halstead Length on an empty code snippet.
    - **Expected Result:** The Halstead Length should be 0.
2. **halsteadLengthOpFaultTest:**
    - **Purpose:** Checks Halstead Length on a code snippet with operator faults.
    - **Expected Result:** The Halstead Length should be 0.
3. **halsteadLengthOpManyTest:**
    - **Purpose:** Checks Halstead Length on a code snippet with many operators.
    - **Expected Result:** The Halstead Length should be 5568.
4. **halsteadLengthOpSingleTest:**
    - **Purpose:** Checks Halstead Length on a code snippet with a single operator.
    - **Expected Result:** The Halstead Length should be 8.
5. **halsteadLengthFaultTest:**
    - **Purpose:** Checks Halstead Length on a code snippet with faults.
    - **Expected Result:** The Halstead Length should be 3420.

Halstead Volume Tests

6. **halsteadVolumeEmptyTest:**

    - **Purpose:** Checks Halstead Volume on an empty code snippet.
    - **Expected Result:** The Halstead Volume should be 0.
7. **halsteadVolumeOpFaultTest:**
    - **Purpose:** Checks Halstead Volume on a code snippet with operator faults.
    - **Expected Result:** The Halstead Volume should be 27.
8. **halsteadVolumeOpManyTest:**
    - **Purpose:** Checks Halstead Volume on a code snippet with many operators.
    - **Expected Result:** The Halstead Volume should be 1302.
9. **halsteadVolumeOpSingleTest:**
    - **Purpose:** Checks Halstead Volume on a code snippet with a single operator.
    - **Expected Result:** The Halstead Volume should be 1.
10. **halsteadVolumeFaultTest:**

- **Purpose:** Checks Halstead Volume on a code snippet with faults.
- **Expected Result:** The Halstead Volume should be 980.

Halstead Difficulty Tests

11. **halsteadDifficultyEmptyTest:**
    - **Purpose:** Checks Halstead Difficulty on an empty code snippet.
    - **Expected Result:** The Halstead Difficulty should be 0.
12. **halsteadDifficultyOpFaultTest:**
    - **Purpose:** Checks Halstead Difficulty on a code snippet with operator faults.
    - **Expected Result:** The Halstead Difficulty should be 0.
13. **halsteadDifficultyOpManyTest:**
    - **Purpose:** Checks Halstead Difficulty on a code snippet with many operators.
    - **Expected Result:** The Halstead Difficulty should be 4.
14. **halsteadDifficultyOpSingleTest:**
    - **Purpose:** Checks Halstead Difficulty on a code snippet with a single operator.
    - **Expected Result:** The Halstead Difficulty should be 0.
15. **halsteadDifficultyFaultTest:**
    - **Purpose:** Checks Halstead Difficulty on a code snippet with faults.
    - **Expected Result:** The Halstead Difficulty should be 2.

Halstead Effort Tests

16. **halsteadEffortEmptyTest:**
    - **Purpose:** Checks Halstead Effort on an empty code snippet.
    - **Expected Result:** The Halstead Effort should be 0.
17. **halsteadEffortOpFaultTest:**

- **Purpose:** Checks Halstead Effort on a code snippet with operator faults.
- **Expected Result:** The Halstead Effort should be 0.

18. **halsteadEffortOpManyTest:**
    - **Purpose:** Checks Halstead Effort on a code snippet with many operators.
    - **Expected Result:** The Halstead Effort should be 5926.
19. **halsteadEffortOpSingleTest:**
    - **Purpose:** Checks Halstead Effort on a code snippet with a single operator.
    - **Expected Result:** The Halstead Effort should be 13.
20. **halsteadEffortFaultTest:**
    - **Purpose:** Checks Halstead Effort on a code snippet with faults.
    - **Expected Result:** The Halstead Effort should be 2482.

Halstead Vocabulary Tests

21. **halsteadVocabEmptyTests:**
    - **Purpose:** Checks Halstead Vocabulary on an empty code snippet.
    - **Expected Result:** The Halstead Vocabulary should be 0.
22. **halsteadVocabularyOpFaultTest:**
    - **Purpose:** Checks Halstead Vocabulary on a code snippet with operator faults.
    - **Expected Result:** The Halstead Vocabulary should be 0.
23. **halsteadVocabularyOpManyTest:**
    - **Purpose:** Checks Halstead Vocabulary on a code snippet with many operators.
    - **Expected Result:** The Halstead Vocabulary should be 164.
24. **halsteadVocabularyOpSingleTest:**
    - **Purpose:** Checks Halstead Vocabulary on a code snippet with a single operator.
    - **Expected Result:** The Halstead Vocabulary should be 9.
25. **halsteadVocabularyFaultTest:**
    - **Purpose:** Checks Halstead Vocabulary on a code snippet with faults.
    - **Expected Result:** The Halstead Vocabulary should be 128.

Operator Tests:

operatorsEmptyTest:

- **Purpose:** Check the operator-related code for an empty code file.
- **Expected Result:** Halstead Length Check should yield 0.

operatorsManyTest:

- **Purpose:** Check the operator-related code for a tree with multiple operators.
- **Expected Result:** Halstead Length Check should yield 22.

operatorsSingleTest:

- **Purpose:** Check the operator-related code for a tree with a single operator.
- **Expected Result:** Halstead Length Check should yield 1.

operatorsFaultTest:

- **Purpose:** Check the operator-related code for a tree with a fault.
- **Expected Result:** Halstead Length Check should yield 0.

Operand Tests:

operandsEmptyTest:

- **Purpose:** Check the operand-related code for an empty code file.
- **Expected Result:** Halstead Length Check should yield 0.

operandsManyTest:

- **Purpose:** Check the operand-related code for a tree with multiple operands.
- **Expected Result:** Halstead Length Check should yield 116.

operandsSingleTest:

- **Purpose:** Check the operand-related code for a tree with a single operand.
- **Expected Result:** Halstead Length Check should yield 1.

operandsFaultTest:

- **Purpose:** Check the operand-related code for a tree with a fault.
- **Expected Result:** Halstead Length Check should yield 7.

Number of Comments Tests:

commentsEmptyTest:

- **Purpose:** Check the comment-related code for an empty code file.
- **Expected Result:** Halstead Length Check should yield 0.

commentsMultiTest:

- **Purpose:** Check the comment-related code for a tree with multiple comments.

- **Expected Result:** Halstead Length Check should yield 2.

commentsBothTest:

- **Purpose:** Check the comment-related code for a tree with both comments and code.
- **Expected Result:** Halstead Length Check should yield 3.

commentsFaultTest:

- **Purpose:** Check the comment-related code for a tree with a fault.
- **Expected Result:** Halstead Length Check should yield 4.

commentsSingleTest:

- **Purpose:** Check the comment-related code for a tree with a single comment.
- **Expected Result:** Halstead Length Check should yield 2.

Lines of Comments Tests:

commentsLinesEmptyTest:

- **Purpose:** Check the lines of comments in code with no comments.
- **Expected Result:** Halstead Length Check should yield 0.

commentsLinesMultiTest:

- **Purpose:** Check the lines of comments in code with multiple comments.
- **Expected Result:** Halstead Length Check should yield 0.

commentsLinesBothTest:

- **Purpose:** Check the lines of comments in code with both comments and code.
- **Expected Result:** Halstead Length Check should yield 0.

commentsLinesFaultTest:

- **Purpose:** Check the lines of comments in code with a fault.
- **Expected Result:** Halstead Length Check should yield 0.

commentsLinesSingleTest:

- **Purpose:** Check the lines of comments in code with a single comment.
- **Expected Result:** Halstead Length Check should yield 0.

Number of Loops Tests:

loopsEmptyTest:

- **Purpose:** Check the number of loops in an empty code file.
- **Expected Result:** Halstead Length Check should yield 0.

loopsForTest:

- **Purpose:** Check the number of loops in code with a "for" loop.
- **Expected Result:** Halstead Length Check should yield 1.

loopsDoTest:

- **Purpose:** Check the number of loops in code with a "do-while" loop.
- **Expected Result:** Halstead Length Check should yield 1.

loopsWhileTest:

- **Purpose:** Check the number of loops in code with a "while" loop.
- **Expected Result:** Halstead Length Check should yield 1.

loopsNestedTest:

- **Purpose:** Check the number of loops in code with nested loops.
- **Expected Result:** Halstead Length Check should yield 9.

loopsFaultedTest:

- **Purpose:** Check the number of loops in code with a fault.
- **Expected Result:** Halstead Length Check should yield 0.

Number of Expressions Tests:

expressionEmptyTest:

- **Purpose:** Check the number of expressions in an empty code file.
- **Expected Result:** Halstead Length Check should yield 0.

expressionTypicalTest:

- **Purpose:** Check the number of expressions in typical code.
- **Expected Result:** Halstead Length Check should yield 99.

expressionFaultTest:

- **Purpose:** Check the number of expressions in code with a fault.
- **Expected Result:** Halstead Length Check should yield 86.

Pit Test results explanation:

Low line coverage shown below means that while full coverage may occur when tests are run on their own with Junit, each individual test doesn't cover a large portion of the code, meaning that there are areas within some tests that won't be covered by that test itself, even though it is covered by the file as a whole.

Low Mutation Kill rate indicates that while they may work for overall functionality, they may not capture subtle changes in the code we are testing.

A test rate averaging across tests of 50 percent indicates that we are catching some mutation errors, but not all. We would have to make each white box test cover more than just a single method in order to have better coverage, and have a better grasp on whether the code is working correctly or was modified in a way that hurts functionality or flow.

Pitclipse was run on the following java files:

(Results are in order of tests given above) :

HalsteadDifficultyWhiteTest.java
HalsteadEffortWhiteTest.java
HalsteadLengthWhiteTest.java
HalsteadVocabularyWhiteTest.java
HalsteadVolumeWhiteTest.java
NumberOfCommentsWhiteTest.java
NumberOfExpressionsWhiteTest.java
NumberOfLinesOfCommentsWhiteTest.java
NumberOfLoopingStatementsWhiteTest.java
NumberOfOperandsWhiteTest.java
NumberOfOperatorsWhiteTest.java

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 28/835 (3%)
>> Generated 173 mutations Killed 4 (2%)
>> Mutations with no coverage 165. Test strength 50%
>> Ran 12 tests (0.07 tests per mutation)
```

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 113/835 (14%)
>> Generated 173 mutations Killed 6 (3%)
>> Mutations with no coverage 160. Test strength 46%
>> Ran 24 tests (0.14 tests per mutation)
```

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 117/835 (14%)
>> Generated 173 mutations Killed 7 (4%)
>> Mutations with no coverage 156. Test strength 41%
>> Ran 34 tests (0.2 tests per mutation)
```

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 121/835 (14%)
>> Generated 173 mutations Killed 7 (4%)
>> Mutations with no coverage 150. Test strength 30%
>> Ran 46 tests (0.27 tests per mutation)
```

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 17/835 (2%)
>> Generated 173 mutations Killed 4 (2%)
>> Mutations with no coverage 165. Test strength 50%
>> Ran 12 tests (0.07 tests per mutation)
```

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 84/835 (10%)
>> Generated 173 mutations Killed 4 (2%)
>> Mutations with no coverage 165. Test strength 50%
>> Ran 12 tests (0.07 tests per mutation)
```

```
============================================================================================
- Statistics
============================================================================================
>> Line Coverage: 25/835 (3%)
>> Generated 173 mutations Killed 6 (3%)
>> Mutations with no coverage 159. Test strength 43%
>> Ran 22 tests (0.13 tests per mutation)
```

```
================================================================================
- Statistics
================================================================================
>> Line Coverage: 28/835 (3%)
>> Generated 173 mutations Killed 4 (2%)
>> Mutations with no coverage 165. Test strength 50%
>> Ran 12 tests (0.07 tests per mutation)
```

Please scroll further down to see pit mutation results

If we were class testing:

If we were class testing rather

```
--------------------------------------------------------------------------------
================================================================================
- Statistics
================================================================================
>> Line Coverage: 46/835 (6%)
>> Generated 173 mutations Killed 4 (2%)
>> Mutations with no coverage 165. Test strength 50%
>> Ran 12 tests (0.07 tests per mutation)
```

```
================================================================================
- Statistics
================================================================================
>> Line Coverage: 120/835 (14%)
>> Generated 173 mutations Killed 7 (4%)
>> Mutations with no coverage 154. Test strength 37%
>> Ran 38 tests (0.22 tests per mutation)
```

```
================================================================================
- Statistics
================================================================================
>> Line Coverage: 129/835 (15%)
>> Generated 173 mutations Killed 7 (4%)
>> Mutations with no coverage 145. Test strength 25%
>> Ran 67 tests (0.39 tests per mutation)
```