Alexander Crowther
Cpt_S 422
Milestone 2

Test Cases

For the project I grouped the tests into categories: CommentsCheckTests, HalsteadCheckTests (expression checks are grouped here), and LoopingCheckTests. For the tests I looked at checking individual components and that basic component functioning, such as getting acceptable tokens, all function correctly.

CommentsCheckTests: With comments I checked that the checkstyle initializes the counters to zero. I then compared the checkstyles get acceptable/default/required tokens match with the expected tokens for comments. I tested comment counting in different ways (testSetCommentsAndLineCounts, testVisitTokenOneComment, testVisitTokenMultipleComments). I verify the results by testing for the expected return and testing that it does not return edge results ($n + 1$ or $n - 1$). These tests are checking the single line and block comments.

HalsteadCheckTests: In addition to verifying the acceptable/default/required tokens match the expected tokens for the checks, I further tested for operators, operands, unique operators, unique operands, and expression tokens to verify that Halstead checkstyle can differentiate the tokens correctly. Next, I tested for the Halstead length, vocabulary, volume, difficulty, and effort formulas return the expected results.

I took the liberty in assuming that answers to the second decimal place is accurate enough for the purpose of these tests. I then tested the isOperator/isOperand 'if' statements (return true/false) by running various tokens through the methods. To test the void VisitTokenWhiteBox method I incrementally added tokens then checked by using the getters for the results. By doing this incrementally I can test each if/else statement. For example, I added a TokenTypes.DIV once, then checked the getOperator and getUniqueOperator for the results. Then I added another TokenTypes.DIV and check the the getOperator and getUniqueOperator again. I repeat this for all the if/else statements for VisitToken.

LoopingChecckTests: These checks are looking for 'while', 'do/while', and 'for' loops. I test that the addLoopStatementCount increments correctly. I also test the isLoopingStatement method by adding different tokens to the VisitToken method.

Test Results

During testing I found that I to fix some of the Halstead formulas as they return wildly off results than expected. I also had to change the integer values to double values specifically for said formulas (e.g. Halstead difficulty has three integer inputs and returns a double).

Initially my Halstead checkstyle was not counting unique operators/operands correctly. I initially built the 'if/else' statements flat. It was having trouble tracking the unique operators/operands. I then imbedded the unique 'if' statement into the operator/operand 'if/else' statements:

```
if (isOperator(ast.getType())) {
        addOperatorsCount();
        if (!uniqueOperatorList.contains(ast.getType())) {
                uniqueOperatorList.add(ast.getType());
                setUniqueOperatorsCount(uniqueOperatorList.size());
        }
} // code continues…
```

Embedding the unique checks corrected this problem nicely.

For Next Time

To reduce code redundancy, I employed @BeforeEach in my tests to setup the mockAST. I later found that while this can work, it can cause testing issues across the tests. Moving forward, it is best to setup the mockAST within each test to ensure that it is setup correctly to the test in question. I also did not test for certain 'exceptions' that can cause incorrect results. For example, is '-3' a number by itself, or the operator '-' and number '3'. I also need to consider/research other common considerations for testing (e.g. the cases when comments would get skipped). I did not test for such outliers, will need to include them.