Deliverable 3 Report

Logan Bach
Cpts 422

1. Code and Test Explanation

See git README.md

2. Test Evaluation

Whitebox Coverage

main.java	95.5 %	1,308	62	1,370
AntiHungarianCheck.java	26.2 %	22	62	84
CommentCheck.java	100.0 %	45	0	45
ExpressionCheck.java	100.0 %	39	0	39
HalsteadLengthCheck.java	100.0 %	327	0	327
HalsteadVolumeCheck.java	100.0 %	386	0	386
LineCommentCheck.java	100.0 %	41	0	41
LoopingStatementCheck.java	100.0 %	47	0	47
OperandCheck.java	100.0 %	75	0	75
OperatorCheck.java	100.0 %	287	0	287
SemiColonCheck.java	100.0 %	39	0	39

Full branch and test coverage for all checks except for the optional AntiHungarianCheck.

Pitclipse Mutation Tests (before improvement)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
10	95%	261/274	61%	41/67	77%	41/53

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
main.java	10	95%	261/274	61%	41/67	77%	41/53

Report generated by PIT 1.6.8

Fault Models

1. CommentCheck Fault Model

Likely Faults

1.1 Block comments counted as two separate comments. **/* example */**
1.2 Counting comment notation inside of comments as comments.
For example, **// /* example */** or ***/ // example */**
1.3 String content miscounts.
Counting **string s = "//example"** or **string s = "/* example/*"**

2. LineCommentCheck Fault Model

Likely Faults

2.1 String content miscounts. Counting 'string s = "//example"' as a comment.
2.2 Double counting comments. The following as two comments '// // example '
2.3 Ignoring comments trailing code on the same line. 'Int x = 0 // example'
2.4 Counting block comments contained to a single line. '/* example */'

3. LoopingStatementCheck Fault Model

Likely Faults

3.1 Loops inside comments being counted
3.2 Loops inside strings being counted
3.3 Loops nested in other loops not being counted correctly

4. OperatorCheck Fault Model

Likely Faults

4.1 Operators inside strings being counted
4.2 Operators inside comments being counted
4.3 Symbols of a pair being double counted.
The following counting as two, **{}**

5. OperandCheck Fault Model

Likely Faults

5.1 Operands inside strings being counted.
5.2 Operands inside comments being counted.
5.3 Operators counted as operands.

6. ExpressionCheck Fault Model

    Likely Faults

    6.1 expressions inside strings being counted.
    6.2 expressions inside comments being counted.
    6.3 expressions inside of method arguments ignored.

7. HalsteadLengthCheck Fault Model

    Likely Faults

    7.1 Operators or operands in string contents are counted.
    7.2 Operators or operands inside comments being counted.
    7.3 Halstead Length being miscalculated.
    7.4 Double counting pairs such as **{}**

8. HalsteadVolumeCheck Fault Model

    8.1 Operators or Operands inside strings being counted.
    8.2 Operators or Operands inside comments being counted.
    8.3 Double counting pairs such as **{}**
    8.4 Not correctly calculating Vocabulary. Duplicates counting as distinct occurrences
    8.5 Program length not calculated correctly.
    8.6 Calculation failure due to Logarithm equaling infinity on zero.

3. Test improvement

WhiteBox Testing

statement/branch coverage is already complete so no need for improving.

Improved Mutation Results

Pit Test Coverage Report

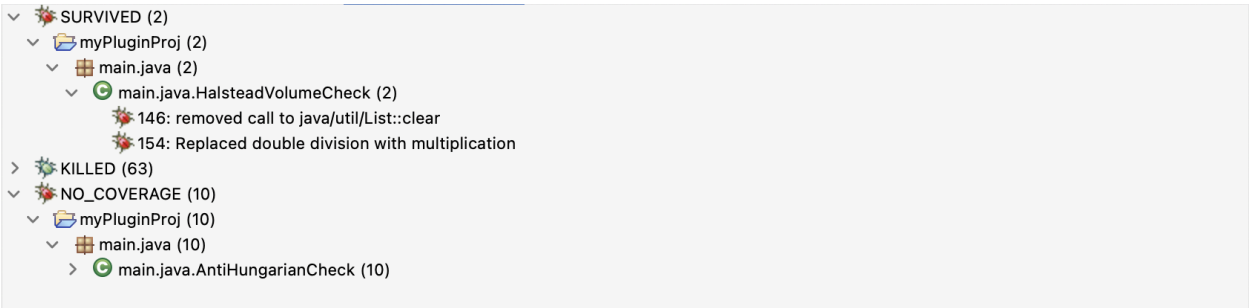
Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
10	96%	292/304	82%	63/77	97%	63/65

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
main.java	10	96%	292/304	82%	63/77	97%	63/65

Report generated by PIT 1.6.8



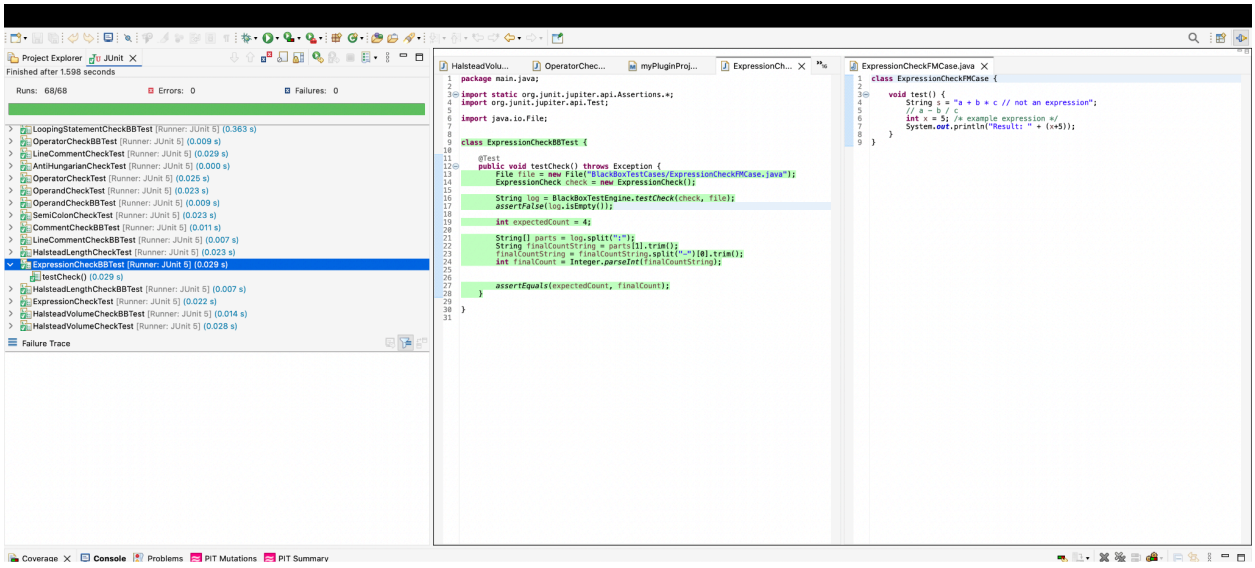
Despite 2 survived bugs in HalsteadVolumeCheck tests the rest are in the optional AntiHungarianCheck.

4. Class Testing

- Class testing could allow for more functionally similar tests to how the checkstyle checks actually run. There would be fewer test cases because the tests would evaluate behavior throughout the class rather than individual methods. Scenarios could be tested where multiple method calls are used to show how they work with one another.

5. ScreenShot of BlackBox in Eclipse

Also the black box engine works through two classes located in the tests folder. The BlackBoxEngineHelper.java and BlackBoxTestEngine.java


Project Explorer
JUnit
Finished after 1.598 seconds
Runs: 68/68
Errors: 0
Failures: 0
LoopingStatementCheckBBTest [Runner: JUnit 5] (0.363 s)
OperatorCheckBBTest [Runner: JUnit 5] (0.009 s)
LineCommentCheckTest [Runner: JUnit 5] (0.029 s)
AntiHungarianCheckTest [Runner: JUnit 5] (0.000 s)
OperatorCheckTest [Runner: JUnit 5] (0.025 s)
OperandCheckTest [Runner: JUnit 5] (0.023 s)
OperandCheckBBTest [Runner: JUnit 5] (0.009 s)
SemiColonCheckTest [Runner: JUnit 5] (0.023 s)
CommentCheckBBTest [Runner: JUnit 5] (0.011 s)
LineCommentCheckBBTest [Runner: JUnit 5] (0.007 s)
HalsteadLengthCheckTest [Runner: JUnit 5] (0.023 s)
ExpressionCheckBBTest [Runner: JUnit 5] (0.029 s)
testCheck() (0.029 s)
HalsteadLengthCheckBBTest [Runner: JUnit 5] (0.007 s)
ExpressionCheckTest [Runner: JUnit 5] (0.022 s)
HalsteadVolumeCheckBBTest [Runner: JUnit 5] (0.014 s)
HalsteadVolumeCheckTest [Runner: JUnit 5] (0.028 s)
Failure Trace
HalsteadVolu...
OperatorChec...
myPluginProj...
ExpressionCh...
package main.java;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.io.File;
class ExpressionCheckBBTest {
@Test
public void testCheck() throws Exception {
File file = new File("BlackBoxTestCases/ExpressionCheckFMCase.java");
ExpressionCheck check = new ExpressionCheck();
String log = BlackBoxTestEngine.testCheck(check, file);
assertFalse(log.isEmpty());
int expectedCount = 4;
String[] parts = log.split("=");
String finalCountString = parts[1].trim();
finalCountString = finalCountString.split("\.")[0].trim();
int finalCount = Integer.parseInt(finalCountString);
assertEquals(expectedCount, finalCount);
}
}
ExpressionCheckFMCase.java
class ExpressionCheckFMCase {
void test() {
String s = "a + b + c // not an expression";
// a - b / c
int x = 5; /* example expression */
System.out.println("Result: " + (x+5));
}
}
Coverage
Console
Problems
PIT Mutations
PIT Summary