Cpt S 422 - Deliverable 2 Test Report

Student: Abylay Dospayev
Project: Custom Checkstyle Checks (Comment, Loop, Operator metrics)

1. Overview
This report documents the unit tests written for three custom Checkstyle checks:
- CommentCountCheck: counts the total number of comments.
- LoopCountCheck: counts the total number of looping statements (for, while, do-while).
- OperatorCountCheck: counts the total number of operators (as defined in the README/spec).

The goal was to achieve high statement and branch coverage and to verify that the metrics are computed correctly under different AST token scenarios.

2. CommentCountCheck Tests
Implemented tests:
- defaultAndAcceptableTokensAreComments():
  Verifies that getDefaultTokens() and getAcceptableTokens() return SINGLE_LINE_COMMENT and BLOCK_COMMENT_BEGIN, and that getRequiredTokens() returns an empty array.
- countsSingleAndBlockCommentsAndResetsAfterFinish():
  Mocks three comment tokens (two single-line and one block).
  Calls visitToken(...) for all of them and uses reflection to verify that internal commentCount becomes 3 before finishTree().
  Also checks that finishTree(...) resets the internal counter back to 0.
- noCommentsLeavesCountZero():
  Calls finishTree(...) directly with no comment tokens encountered and checks that commentCount remains 0.

Coverage for CommentCountCheck.java:
- Overall line coverage: 80.0% (36 / 45 instructions).
- getDefaultTokens(): 100% coverage (11 / 11).
- getAcceptableTokens(): 100% coverage (3 / 3).
- getRequiredTokens(): 100% coverage (3 / 3).
- visitToken(DetailAST): 100% coverage (7 / 7).
- finishTree(DetailAST): 40% coverage (6 / 15).

3. LoopCountCheck Tests
Implemented tests:
- defaultAndAcceptableTokensAreLoops():
  Verifies that the default and acceptable tokens are LITERAL_FOR, LITERAL_WHILE, and LITERAL_DO, and that required tokens is an empty array.
- noLoopsKeepsCountZeroAndResetInFinishTree():
  Calls beginTree(...) and finishTree(...) with no loop tokens and ensures loopCount stays at 0 and is reset correctly.
- countsForAndPlainWhile():
  Mocks a for-loop AST and a while-loop AST whose parent is null (plain while).
  After calling visitToken(...) for both, verifies loopCount == 2 before the reset, and then confirms that finishTree(...) resets loopCount to 0.
- countsDoWhileOnce():
  Mocks a LITERAL_DO token and a LITERAL_WHILE token whose parent is the DO node.

Ensures that the loop is counted exactly once (loopCount == 1) and then reset in finishTree(...).
- ignoresNonLoopTokens():
Mocks a non-loop token (IDENT) and verifies that visitToken(...) does not change the loopCount.

Coverage for LoopCountCheck.java:
- Overall line coverage: 88.6% (70 / 79 instructions).
- beginTree(DetailAST): 100% coverage (4 / 4).
- getDefaultTokens(): 100% coverage (15 / 15).
- getAcceptableTokens(): 100% coverage (3 / 3).
- getRequiredTokens(): 100% coverage (3 / 3).
- visitToken(DetailAST): 100% coverage (33 / 33).
- finishTree(DetailAST): 40% coverage (6 / 15).

4. OperatorCountCheck Tests
Implemented tests:
- defaultAndAcceptableTokensMatch():
Asserts that getDefaultTokens() and getAcceptableTokens() return identical arrays, and that getRequiredTokens() returns an empty array.
- countsMultipleOperatorsAndResets():
Mocks PLUS, MINUS, and ASSIGN tokens. After visitToken(...) on all three, verifies that getOperatorCount() returns 3. After finishTree(...), verifies that the internal operator counter has been reset to 0.
- ignoresNonOperatorTokens():
Mocks an IDENT token (not in the OPERATOR_SET) and verifies that it does not affect the operatorCount.

Coverage for OperatorCountCheck.java:
- Overall line coverage: 95.0% (172 / 181 instructions).
- finishTree(DetailAST): 40% coverage (6 / 15).

5. Overall Project Coverage
Project-wide coverage as reported by Eclipse:
- Project Delivarable1: 96.5% (735 / 762 instructions).
- Package checkStylePackageTest: 91.1% (278 / 305 instructions).

What worked and what didn't:
- All three check classes compiled and all JUnit 5 tests passed successfully.
- Mockito-based mocking of DetailAST tokens allowed us to drive specific branches in visitToken(...) without needing full Java source files.
- We achieved 100% coverage for most methods (token arrays, beginTree, visitToken), but finishTree(DetailAST) in all three checks only reached 40% coverage due to logging and guard logic that is harder to exercise in unit tests.