Structural-Based Testing Assignment

Sumanth Reddy Velagala

School of Computing and Augmented Intelligence, Arizona State University

February 25, 2025

Part 1

Description of the tool and type of coverage

JaCoCo(Java Code Coverage) is a popular tool for Java applications which provides detailed information on the code coverage achieved. Supports testing for Java 7 and 8 and integration with popudoelar Integrated Development Environment(IDE) such as Ant, Maven, Eclipse and Gradle. JaCoCo supports statement coverage, decision coverage, instruction coverage and cyclomatic complexity. (BrowserStack, n.d.)

Description of test cases developed

```
@Test
public void testExactPaymentCandy() {
    assertEquals("Item dispensed.", VendingMachine.dispenseItem(20, "candy"));
}

@Test
public void testExactPaymentCoke() {
    assertEquals("Item dispensed.", VendingMachine.dispenseItem(25, "coke"));
}

@Test
public void testExactPaymentCoffee() {
    assertEquals("Item dispensed.", VendingMachine.dispenseItem(45, "coffee"));
}

@Test
public void testOverpaymentCandy() {
    assertEquals("Item dispensed and change of 30 returned", VendingMachine.dispenseItem(50, "candy"));
}

@Test
public void testInsufficientFundsCandy() {
    assertEquals("Item not dispensed, missing 5 cents. Cannot purchase item.", VendingMachine.dispenseItem(15, "candy"));
}

@Test
public void testInsufficientFundsCoke() {
    assertEquals("Item not dispensed, missing 3 cents. Can purchase candy.", VendingMachine.dispenseItem(22, "coke"));
}

@Test
public void testInsufficientFundsCoffee() {
    assertEquals("Item not dispensed, missing 15 cents. Can purchase candy or coke.", VendingMachine.dispenseItem(30, "coff
}
```

Figure 1 : Snippet of test cases

1. testExactPaymentCandy (20,"candy") - covers 3 statements (cost = 20, change = 0, returnvalue = " Item dispensed") and 2 branches (if item == "candy", else if (input == cost)).

2. testExactPayementCoke(25,"coke") - covers 3 statements (cost = 25, change = 0, returnvalue = " Item dispensed") and 2 branches (if item == "coke", else if (input == cost)).

3. testExactPayementCoffee(45,”coffee”) - covers 3 statements (cost = 45, change = 0, returnvalue = “ Item dispensed”) and 2 branches (if item == “coffee”, else if (input == cost)).

4. testOverPayementCandy(50,”candy”) - covers 3 statements (cost = 20, change = input-change, returnvalue = “ Item dispensed and change of 30 returned”) and 2 branches (if item == “candy”, else if (input > cost)).

5. testInsufficientFundscandy(15,”candy”) - covers 3 statements (cost = 20, change = input-change, returnvalue = “ Item not dispensed, missing 5 cents. Cannot purchase item”) and 5 branches (if item == “candy”, else ,if (input < 45), if(input < 25), if(input(<20)).

6. testInsufficientFundscoke(22,”coke”) - covers 3 statements (cost = 25, change = input-change, returnvalue = “ Item not dispensed, missing 3 cents. Cannot purchase item”) and 4 branches (if item == “coke”, else ,if (input < 45), if(input < 25)).

7. testInsufficientFundscoffee(30,”coffee”) - covers 3 statements (cost = 45, change = input-change, returnvalue = “ Item not dispensed, missing 15 cents. Cannot purchase item”) and 3 branches (if item == “coffee”, else ,if (input < 45)).

Code Coverage

The above test cases achieved 100% statement coverage and 93% of decision coverage missing one decision.

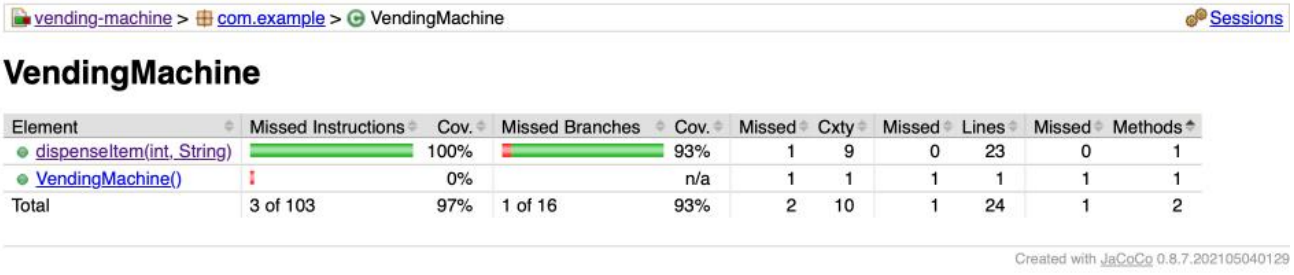

Figure 2 : Snippet of code coverage achieved using JaCoCo

Part 2

Description of the tool and type of analysis

PMD is a static code anlaysis, which can be incorporated into IDE, automation tool like maven by adding dependencies to the pom.xml. PMD provides immediate feedback when there is change in the code using PMD plugin. This tool analyses dead code, performance issues, style issues and dangerous code practices.

Description and anomalies detected by PMD

1. Priority 1 ClassNamingConventions - The name StaticAnalysis doesn't follow the naming convention.

2. Priority 3 UseUtilityClass anamoly detected as all methods are static.

3. Two UnusedLocalVariables were found (weight, length).

4. UseEqualToCompareStrings anomaly - usage of '==' is used instead of equals().

5. ControlStatementBraces anomaly - presence of missing braces.

Last Published: 2025-02-25 | Version: 1.0-SNAPSHOT

Built by Maven

PMD Results

The following document contains the results of PMD 6.21.0.

Violations By Priority

Priority 1

com/example/StaticAnalysis.java

Rule	Violation	Line
ClassNamingConventions	The utility class name 'StaticAnalysis' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'	4–41

Priority 3

com/example/App.java

Rule	Violation	Line
UseUtilityClass	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.	8–13

com/example/StaticAnalysis.java

Rule	Violation	Line
UseUtilityClass	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.	5–41
UnusedLocalVariable	Avoid unused local variables such as 'weight'.	15
UnusedLocalVariable	Avoid unused local variables such as 'length'.	16
UseEqualsToCompareStrings	Use equals() to compare strings instead of '==' or '!='	24
ControlStatementBraces	This statement should have braces	34
ControlStatementBraces	This statement should have braces	36

Figure 3 : Snippet of static analysis achieved using PMD

Files

com/example/App.java

Rule	Violation	Priority	Line
UseUtilityClass	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.	3	8–13

com/example/StaticAnalysis.java

Rule	Violation	Priority	Line
ClassNamingConventions	The utility class name 'StaticAnalysis' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'	1	4–41
UseUtilityClass	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.	3	5–41
UnusedLocalVariable	Avoid unused local variables such as 'weight'.	3	15
UnusedLocalVariable	Avoid unused local variables such as 'length'.	3	16
UseEqualsToCompareStrings	Use equals() to compare strings instead of '==' or '!='	3	24
ControlStatementBraces	This statement should have braces	3	34
ControlStatementBraces	This statement should have braces	3	36

Figure 4 : Snippet of static analysis achieved using PMD

Assessment of tool

Features : Detects wide range of anomalies along with violation priority with clear description of the violation and line number.

Type of anomalies covered : ClassNamingConventions, UseUtilityClass, UnusedLocal Variable, UseEqualsToCompareStrings, ControlStatementBraces.

Ease of Use : can be used directly into automation tools like maven by incorporating dependencies into the pom.xml file with requirement for minimal knowledge on setup.

References

BrowserStack. (n.d.). Code coverage tools. BrowserStack. Retrieved February 25, 2025, from https://www.browserstack.com/guide/code-coverage-tools