Homework #3

Submitted by: Aditi Shashank Joshi (1222838916)

Part 1:

The vast majority of current software projects today use the test-driven development (TDD) methodology. This method requires that you constantly write automated regression tests in addition to your code. If you use the Test First methodology, you even write your test cases before composing any code, which helps you define clear APIs and develop minimal, simple implementations. Code coverage gauges the thoroughness of your test suites, whereas your test suites confirm the accuracy of your code. [1]

- Description of the tool used and the types of coverage it provides:

Tool Chosen - JaCoCo

Code coverage is a software metric used to measure how many lines of our code are executed during automated tests. JaCoCo is a free code coverage library for Java, which has been created by the EclEmma team based on the lessons learned from using and integrating existing libraries for many years.

Some features of JaCoCo are mentioned below [2]:

- Based on byte code, can work in the absence of source code files.
- Custom class loaders can be created through the public API available.
- Supports different JVM-based languages.
- Reports can be generated in different formats - CSV, XML, HTML.

- Minimal runtime overhead with simple integration and good performance for large-scale projects.
- Minimal dependencies on external libraries.

JaCoCo provides various types of coverage [3]:

- **Instruction Coverage**: Provides information about the amount of code that has been executed or missed. This is independent of source formatting and always available, even in the absence of debugging information.
- **Decision Coverage**: Provides the branch coverage of all the if, else and switch statements. It counts the total number of branches in a method and determines the amount of executed or missed branches. Like instruction coverage, this coverage is also always available, even in the absence of debugging information. Exception handling is not considered in branches.

  JaCoCo has multiple decision points. There are:

  - No Coverage: No branches in the line have been executed.
  - Partial Coverage: Partial branches in the line have been executed.
  - Full Coverage: All branches in the line have been executed.
- **Cyclomatic Complexity**: Provides cyclomatic complexity for each non-abstract method. It is the minimum number of paths that has to be taken to generate all possible paths via a method. This measure informs us of the number of unit test cases required to fully cover a certain piece of software. It also calculates covered and missed complexity. This coverage is also always available, even in the absence of debugging information.

- **Class Coverage**: If at least one method in a class is executed during coverage analysis, the class is marked as executed. Constructors and static initializers are also counted as methods of a class in class coverage.
- **Method Coverage**: A method is considered to have been executed when at least one instruction has been executed. Constructors and static initializers are also counted as methods. Non-abstract method contains at least one instruction.
- **Line Coverage**: A source line can have multiple instructions associated with it. If at least one of these instructions is executed, the line is considered to be executed. As there are multiple instructions associated with a source line, there are different levels of coverage - no coverage, partial coverage and full coverage.

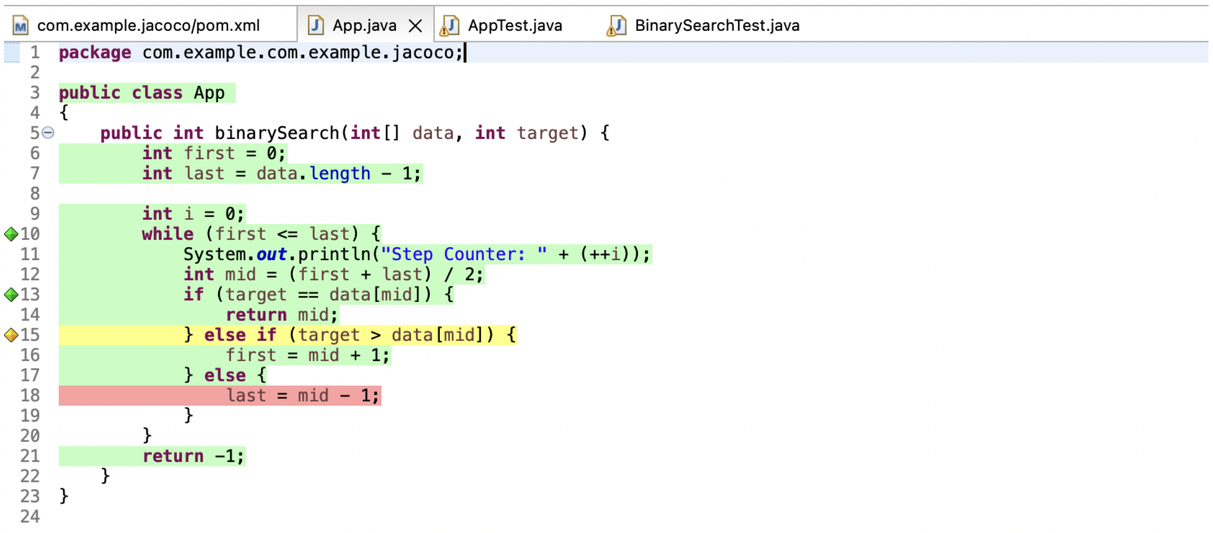
- **<u>Source listing of your code</u>**

```java
package com.example.com.example.jacoco;

public class App
{
    public int binarySearch(int[] data, int target) {
        int first = 0;
        int last = data.length - 1;

        int i = 0;
        while (first <= last) {
            System.out.println("Step Counter: " + (++i));
            int mid = (first + last) / 2;
            if (target == data[mid]) {
                return mid;
            } else if (target > data[mid]) {
                first = mid + 1;
            } else {
                last = mid - 1;
            }
        }
        return -1;
    }
}
```

- **Your test cases (both sets)**

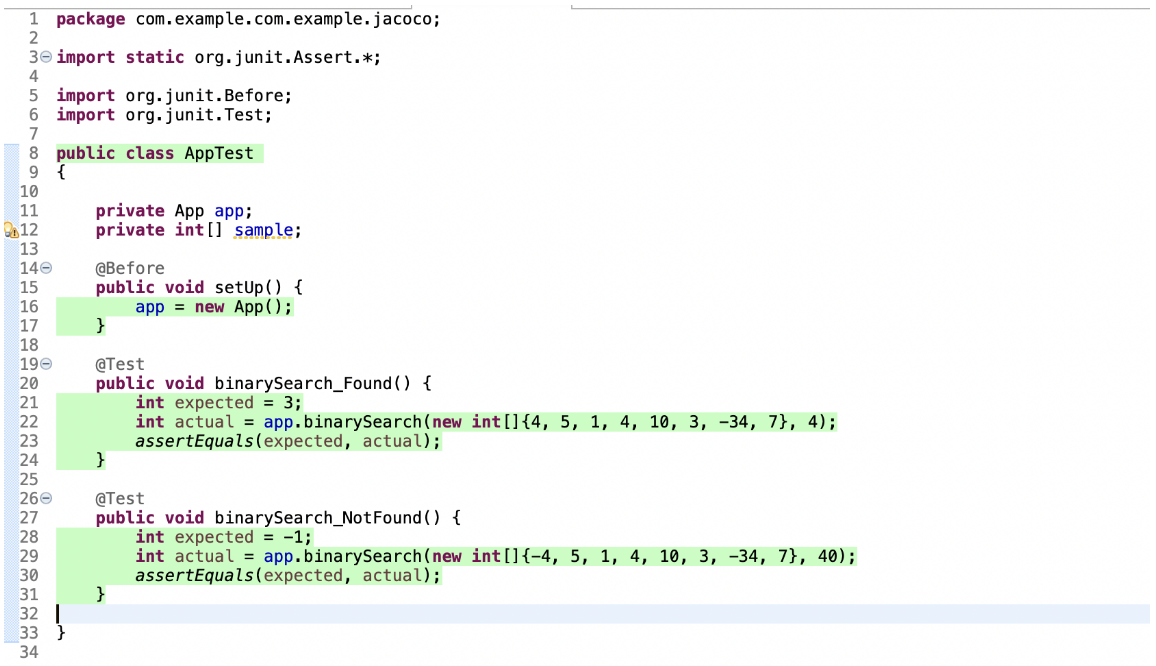
1. First Set

Code:



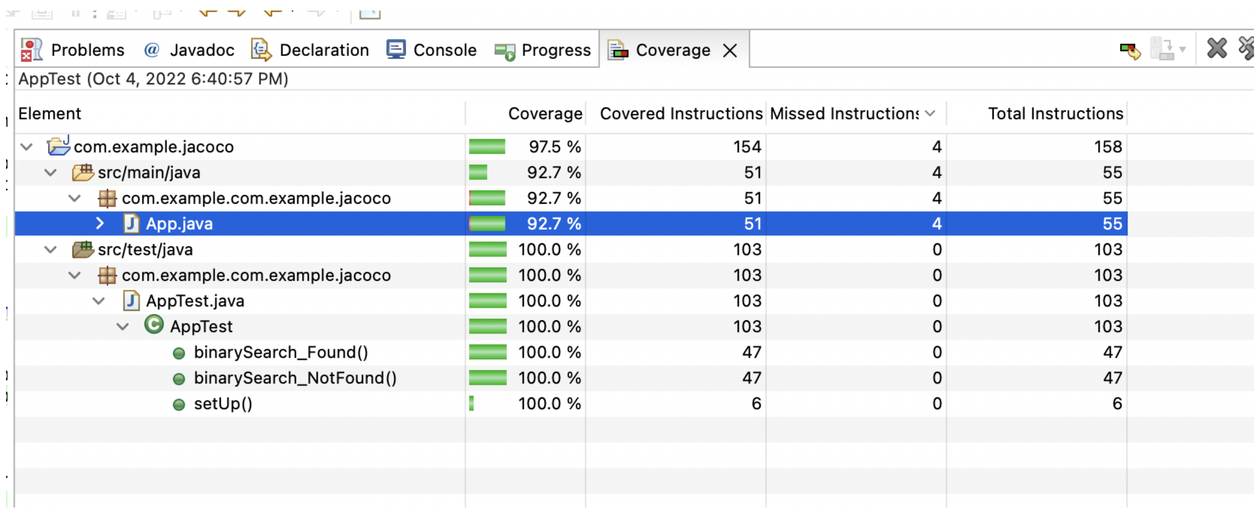
```java
package com.example.com.example.jacoco;

public class App
{
    public int binarySearch(int[] data, int target) {
        int first = 0;
        int last = data.length - 1;

        int i = 0;
        while (first <= last) {
            System.out.println("Step Counter: " + (++i));
            int mid = (first + last) / 2;
            if (target == data[mid]) {
                return mid;
            } else if (target > data[mid]) {
                first = mid + 1;
            } else {
                last = mid - 1;
            }
        }
        return -1;
    }
}
```

Test cases:



```java
package com.example.com.example.jacoco;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class AppTest
{

    private App app;
    private int[] sample;

    @Before
    public void setUp() {
        app = new App();
    }

    @Test
    public void binarySearch_Found() {
        int expected = 3;
        int actual = app.binarySearch(new int[]{4, 5, 1, 4, 10, 3, -34, 7}, 4);
        assertEquals(expected, actual);
    }

    @Test
    public void binarySearch_NotFound() {
        int expected = -1;
        int actual = app.binarySearch(new int[]{-4, 5, 1, 4, 10, 3, -34, 7}, 40);
        assertEquals(expected, actual);
    }

}
```

Coverage Report:

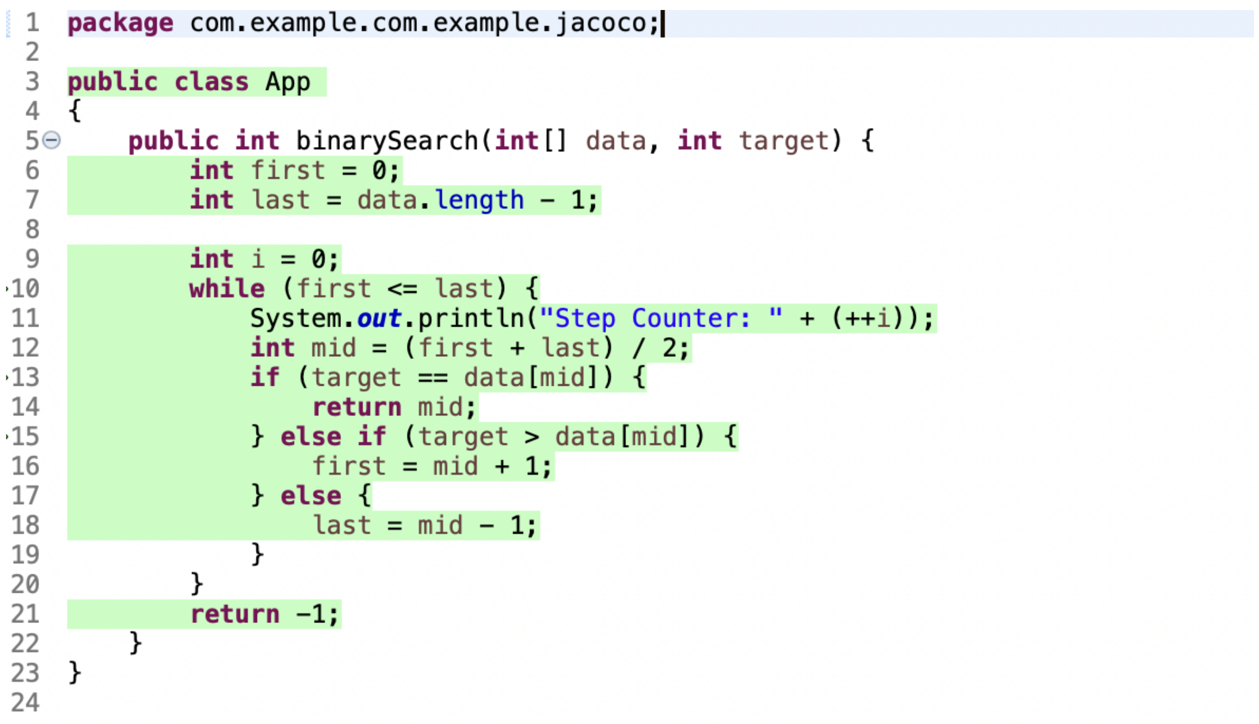


AppTest (Oct 4, 2022 6:40:57 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
com.example.jacoco	97.5 %	154	4	158
src/main/java	92.7 %	51	4	55
com.example.com.example.jacoco	92.7 %	51	4	55
App.java	92.7 %	51	4	55
src/test/java	100.0 %	103	0	103
com.example.com.example.jacoco	100.0 %	103	0	103
AppTest.java	100.0 %	103	0	103
AppTest	100.0 %	103	0	103
binarySearch_Found()	100.0 %	47	0	47
binarySearch_NotFound()	100.0 %	47	0	47
setUp()	100.0 %	6	0	6

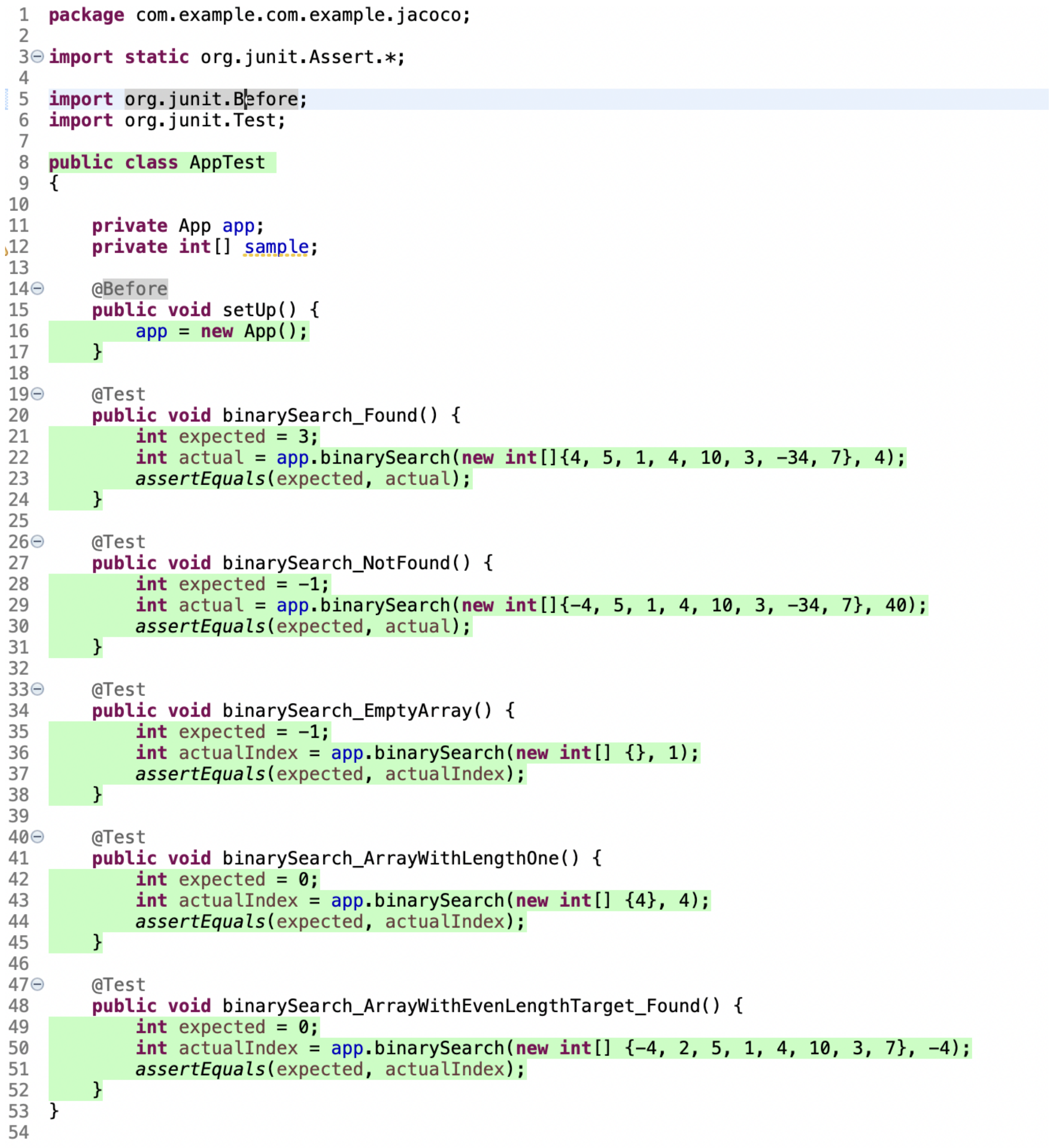
2. Second Set

Code



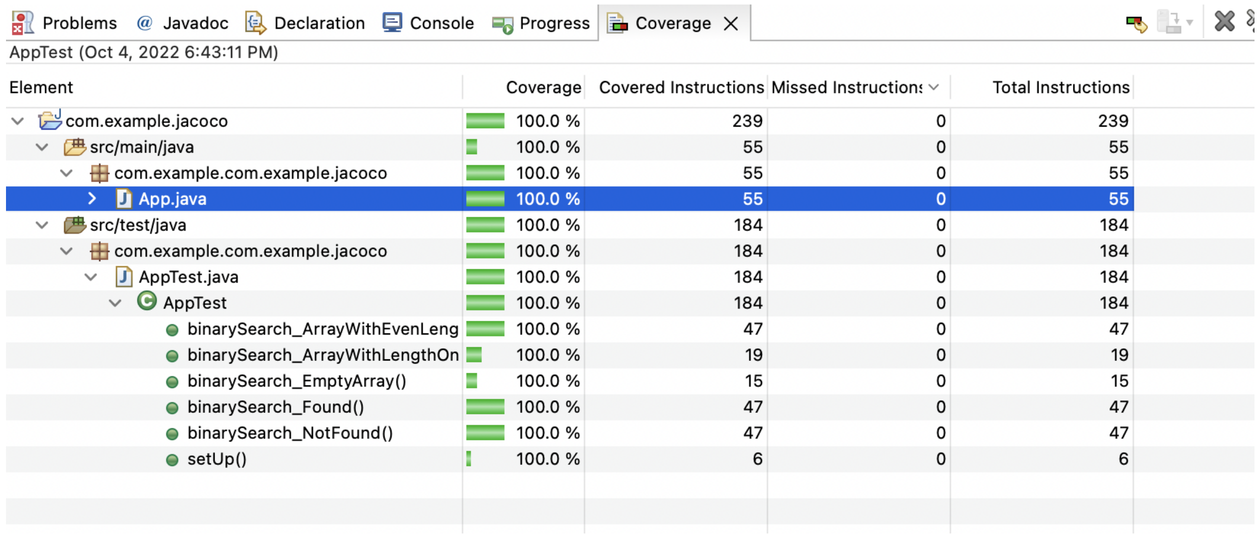
```java
package com.example.com.example.jacoco;

public class App
{
    public int binarySearch(int[] data, int target) {
        int first = 0;
        int last = data.length - 1;

        int i = 0;
        while (first <= last) {
            System.out.println("Step Counter: " + (++i));
            int mid = (first + last) / 2;
            if (target == data[mid]) {
                return mid;
            } else if (target > data[mid]) {
                first = mid + 1;
            } else {
                last = mid - 1;
            }
        }
        return -1;
    }
}
```

Test cases:



```java
package com.example.com.example.jacoco;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class AppTest
{

    private App app;
    private int[] sample;

    @Before
    public void setUp() {
        app = new App();
    }

    @Test
    public void binarySearch_Found() {
        int expected = 3;
        int actual = app.binarySearch(new int[]{4, 5, 1, 4, 10, 3, -34, 7}, 4);
        assertEquals(expected, actual);
    }

    @Test
    public void binarySearch_NotFound() {
        int expected = -1;
        int actual = app.binarySearch(new int[]{-4, 5, 1, 4, 10, 3, -34, 7}, 40);
        assertEquals(expected, actual);
    }

    @Test
    public void binarySearch_EmptyArray() {
        int expected = -1;
        int actualIndex = app.binarySearch(new int[] {}, 1);
        assertEquals(expected, actualIndex);
    }

    @Test
    public void binarySearch_ArrayWithLengthOne() {
        int expected = 0;
        int actualIndex = app.binarySearch(new int[] {4}, 4);
        assertEquals(expected, actualIndex);
    }

    @Test
    public void binarySearch_ArrayWithEvenLengthTarget_Found() {
        int expected = 0;
        int actualIndex = app.binarySearch(new int[] {-4, 2, 5, 1, 4, 10, 3, 7}, -4);
        assertEquals(expected, actualIndex);
    }
}
```

Coverage Report:



AppTest (Oct 4, 2022 6:43:11 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
com.example.jacoco	100.0 %	239	0	239
src/main/java	100.0 %	55	0	55
com.example.com.example.jacoco	100.0 %	55	0	55
App.java	100.0 %	55	0	55
src/test/java	100.0 %	184	0	184
com.example.com.example.jacoco	100.0 %	184	0	184
AppTest.java	100.0 %	184	0	184
AppTest	100.0 %	184	0	184
binarySearch_ArrayWithEvenLeng	100.0 %	47	0	47
binarySearch_ArrayWithLengthOn	100.0 %	19	0	19
binarySearch_EmptyArray()	100.0 %	15	0	15
binarySearch_Found()	100.0 %	47	0	47
binarySearch_NotFound()	100.0 %	47	0	47
setUp()	100.0 %	6	0	6

- **<u>Your evaluation of the tool's usefulness</u>**
    - Different levels (no, partial, full coverage) of decision/branch coverage help developers and testers understand to what extent branches have been executed.
    - Can easily be integrated with build tools (Maven, Gradle, Ant) and IDEs (IDEA, Eclipse).
    - The exact location of line, class, method is highlighted to show the coverage extent, thereby, saving time for developers and testers.
    - Takes minimal time to execute and generate reports, performed at the same time as unit testing.
    - Lightweight and has minimal dependencies on other external libraries.
    - Reports can be generated in different formats - CSV, XML or HTML.
    - Independent of languages; works for any language that is based on JVM.
    - This tool is still updated and maintained regularly, unlike EMMA and Cobertura.
    - The tool is really useful, user-friendly and configurable based on public APIs available. It is regularly updated and maintained, thereby, making it an industry-standard coverage testing tool.

Part 2:

- Description of the tool used and the types of coverage it provides:

Tool Chosen - PMD

PMD is an extensible and cross-language static code analyzer. It helps developers adhere to good coding practices and enforces them to write efficient source code. It is highly configurable and can support any coding standard. It can also be configured to support a custom coding standard for a team if required. It can easily be integrated with build tools & IDEs and works for multiple programming languages. [3]

PMD provides multiple types of analysis. Some of these are mentioned below: [4]

- Best Practices: Enforces best practices, Loose Coupling, Unused Imports, Unused Private Field, Use Collection is empty
- Code Style: Enforces a specific coding style, At least one constructor, Duplicate imports, If Else statements must use braces
- Design: Avoid Throwing Null Pointer Exception, Excessive Imports
- Documentation: Comment Required, Comment Size
- Error Prone: Call Super First, Empty Catch Block
- Multithreading: Avoid Thread Group, Avoid Using Volatile
- Performance: Use Arrays As List, Use String Buffer Length
- Security: Hard Coded Crypto Key, Insecure Crypto Iv

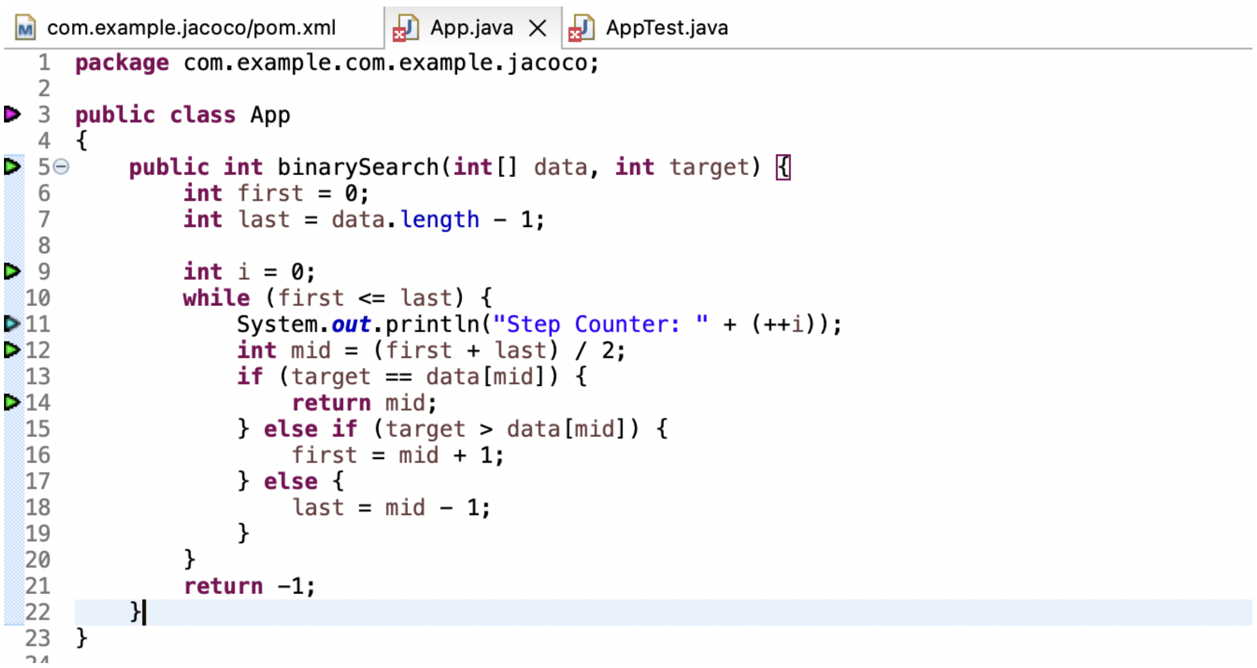
- **<u>Source listing of your code</u>**

```
package com.example.com.example.jacoco;

public class App
{
    public int binarySearch(int[] data, int target) {
        int first = 0;
        int last = data.length - 1;

        int i = 0;
        while (first <= last) {
            System.out.println("Step Counter: " + (++i));
            int mid = (first + last) / 2;
            if (target == data[mid]) {
                return mid;
            } else if (target > data[mid]) {
                first = mid + 1;
            } else {
                last = mid - 1;
            }
        }
        return -1;
    }
}
```

- **<u>Your test cases (both sets)</u>**

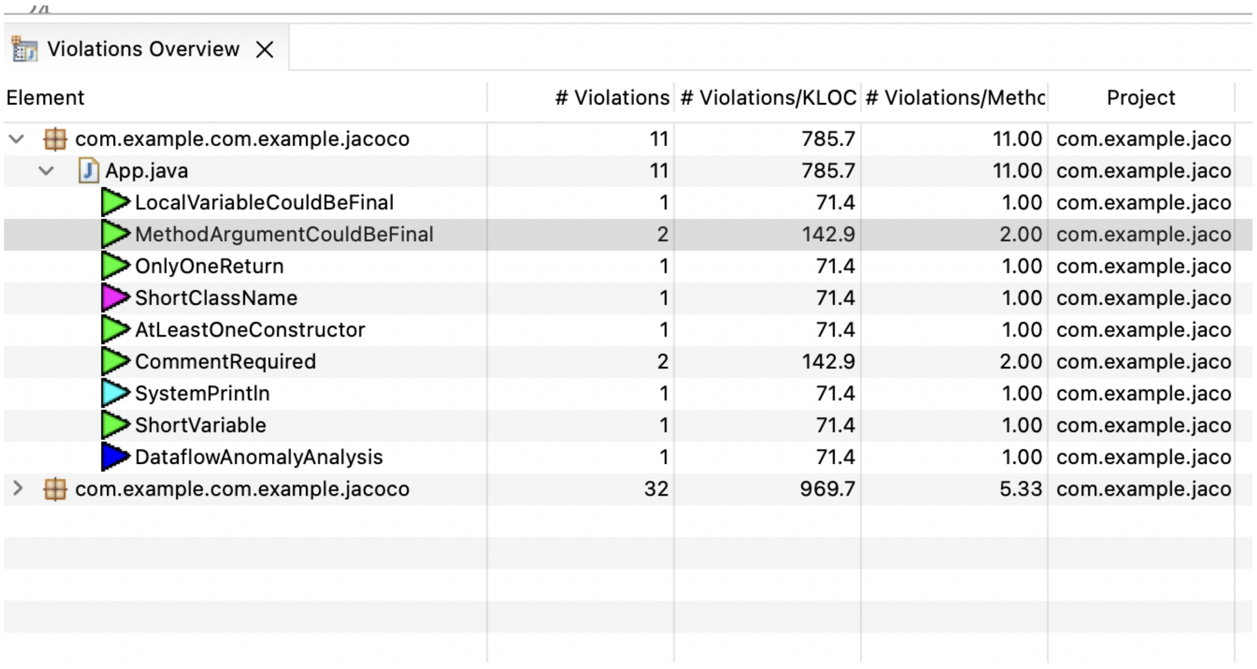
1. First Set - Before adding any anomalies

Code:



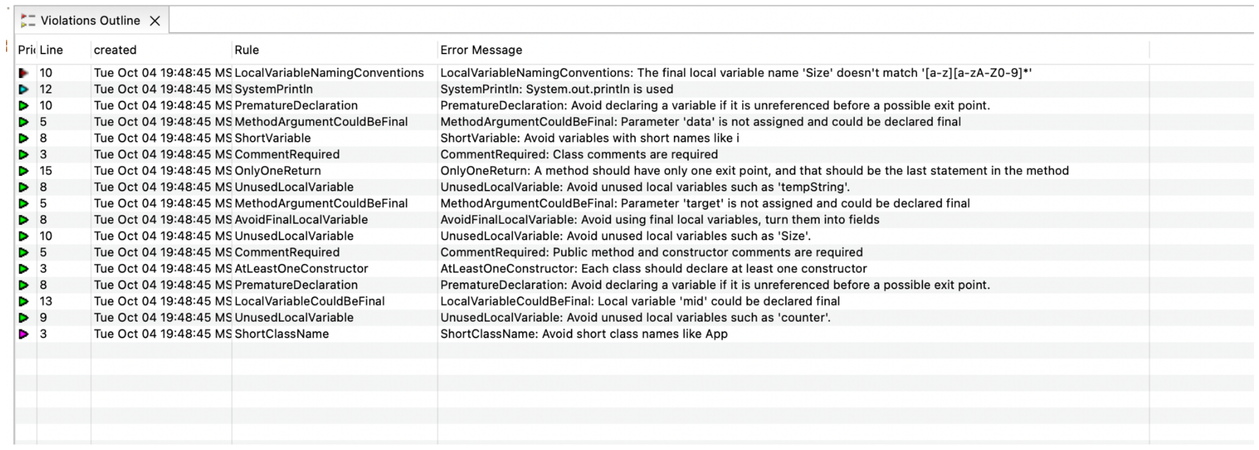
```java
package com.example.com.example.jacoco;

public class App
{
    public int binarySearch(int[] data, int target) {
        int first = 0;
        int last = data.length - 1;

        int i = 0;
        while (first <= last) {
            System.out.println("Step Counter: " + (++i));
            int mid = (first + last) / 2;
            if (target == data[mid]) {
                return mid;
            } else if (target > data[mid]) {
                first = mid + 1;
            } else {
                last = mid - 1;
            }
        }
        return -1;
    }
}
```

Anomalies Overview:



Element	# Violations	# Violations/KLOC	# Violations/Methc	Project
com.example.com.example.jacoco	11	785.7	11.00	com.example.jaco
App.java	11	785.7	11.00	com.example.jaco
LocalVariableCouldBeFinal	1	71.4	1.00	com.example.jaco
MethodArgumentCouldBeFinal	2	142.9	2.00	com.example.jaco
OnlyOneReturn	1	71.4	1.00	com.example.jaco
ShortClassName	1	71.4	1.00	com.example.jaco
AtLeastOneConstructor	1	71.4	1.00	com.example.jaco
CommentRequired	2	142.9	2.00	com.example.jaco
SystemPrintln	1	71.4	1.00	com.example.jaco
ShortVariable	1	71.4	1.00	com.example.jaco
DataflowAnomalyAnalysis	1	71.4	1.00	com.example.jaco
com.example.com.example.jacoco	32	969.7	5.33	com.example.jaco

Violations Outline:

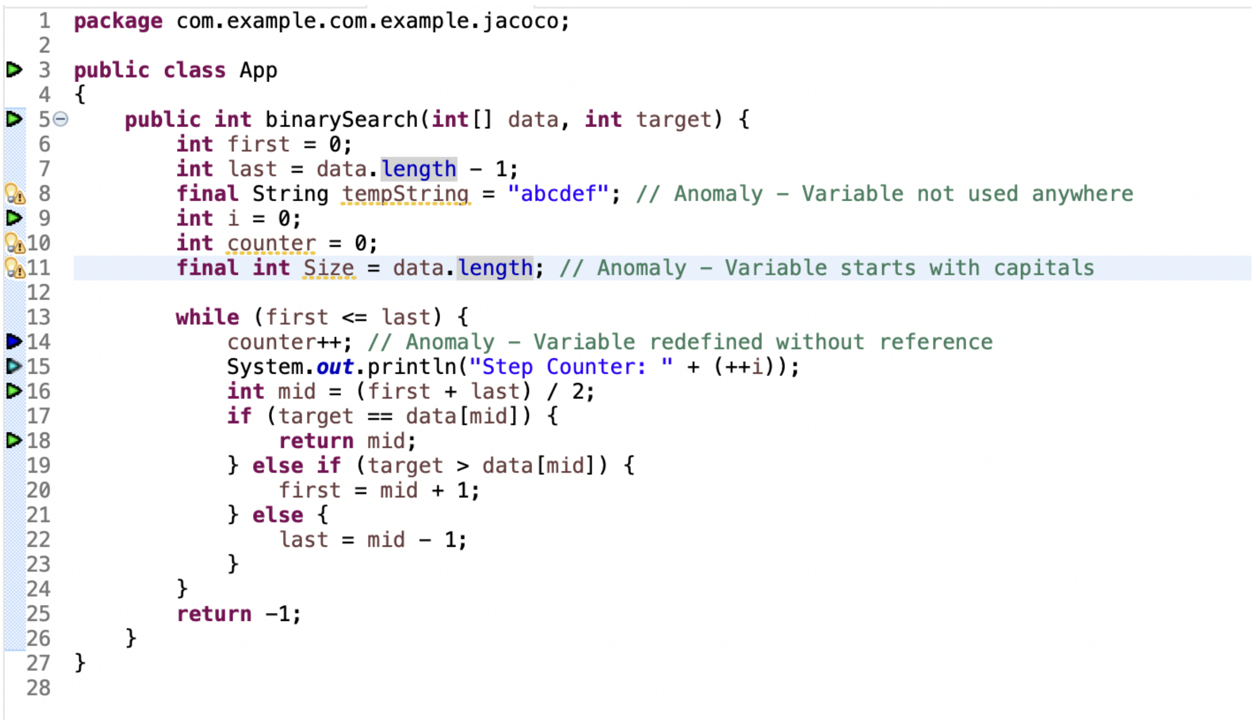


Violations Outline

Pri	Line	created	Rule	Error Message
▶	10	Tue Oct 04 19:48:45 MS	LocalVariableNamingConventions	LocalVariableNamingConventions: The final local variable name 'Size' doesn't match '[a-z][a-zA-Z0-9]*'
▶	12	Tue Oct 04 19:48:45 MS	SystemPrintln	SystemPrintln: System.out.println is used
▶	10	Tue Oct 04 19:48:45 MS	PrematureDeclaration	PrematureDeclaration: Avoid declaring a variable if it is unreferenced before a possible exit point.
▶	5	Tue Oct 04 19:48:45 MS	MethodArgumentCouldBeFinal	MethodArgumentCouldBeFinal: Parameter 'data' is not assigned and could be declared final
▶	8	Tue Oct 04 19:48:45 MS	ShortVariable	ShortVariable: Avoid variables with short names like i
▶	3	Tue Oct 04 19:48:45 MS	CommentRequired	CommentRequired: Class comments are required
▶	15	Tue Oct 04 19:48:45 MS	OnlyOneReturn	OnlyOneReturn: A method should have only one exit point, and that should be the last statement in the method
▶	8	Tue Oct 04 19:48:45 MS	UnusedLocalVariable	UnusedLocalVariable: Avoid unused local variables such as 'tempString'.
▶	5	Tue Oct 04 19:48:45 MS	MethodArgumentCouldBeFinal	MethodArgumentCouldBeFinal: Parameter 'target' is not assigned and could be declared final
▶	8	Tue Oct 04 19:48:45 MS	AvoidFinalLocalVariable	AvoidFinalLocalVariable: Avoid using final local variables, turn them into fields
▶	10	Tue Oct 04 19:48:45 MS	UnusedLocalVariable	UnusedLocalVariable: Avoid unused local variables such as 'Size'.
▶	5	Tue Oct 04 19:48:45 MS	CommentRequired	CommentRequired: Public method and constructor comments are required
▶	3	Tue Oct 04 19:48:45 MS	AtLeastOneConstructor	AtLeastOneConstructor: Each class should declare at least one constructor
▶	8	Tue Oct 04 19:48:45 MS	PrematureDeclaration	PrematureDeclaration: Avoid declaring a variable if it is unreferenced before a possible exit point.
▶	13	Tue Oct 04 19:48:45 MS	LocalVariableCouldBeFinal	LocalVariableCouldBeFinal: Local variable 'mid' could be declared final
▶	9	Tue Oct 04 19:48:45 MS	UnusedLocalVariable	UnusedLocalVariable: Avoid unused local variables such as 'counter'.
▶	3	Tue Oct 04 19:48:45 MS	ShortClassName	ShortClassName: Avoid short class names like App

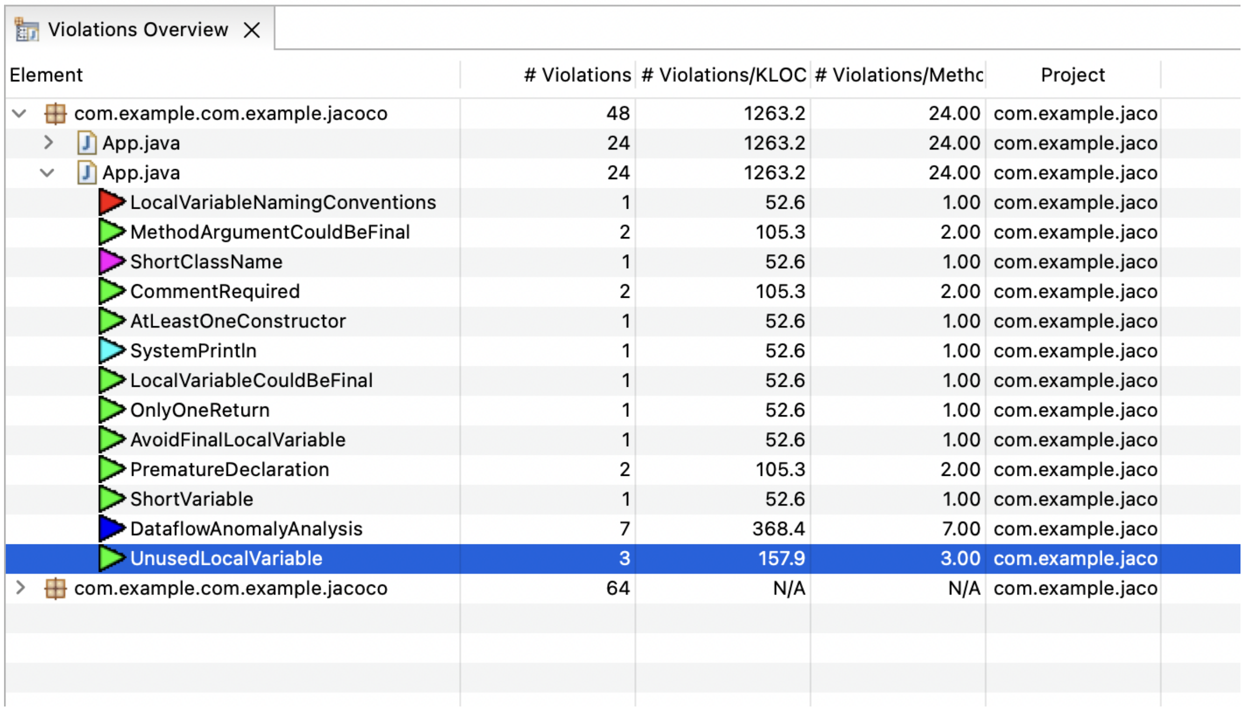
2. Second Set - After adding anomalies

Code



```java
package com.example.com.example.jacoco;

public class App
{
    public int binarySearch(int[] data, int target) {
        int first = 0;
        int last = data.length - 1;
        final String tempString = "abcdef"; // Anomaly - Variable not used anywhere
        int i = 0;
        int counter = 0;
        final int Size = data.length; // Anomaly - Variable starts with capitals

        while (first <= last) {
            counter++; // Anomaly - Variable redefined without reference
            System.out.println("Step Counter: " + (++i));
            int mid = (first + last) / 2;
            if (target == data[mid]) {
                return mid;
            } else if (target > data[mid]) {
                first = mid + 1;
            } else {
                last = mid - 1;
            }
        }
        return -1;
    }
}
```

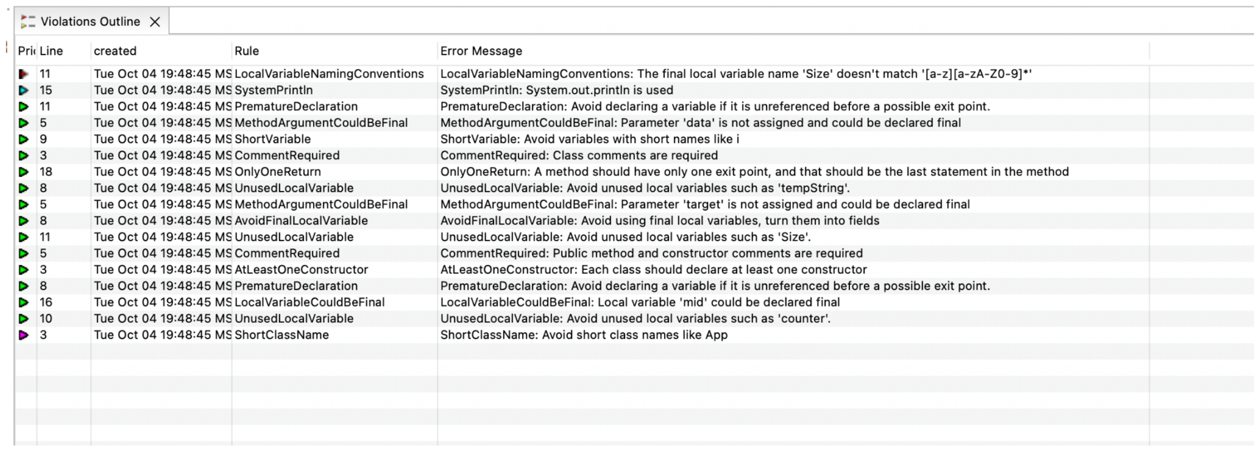
Anomalies Overview:



Violations Overview

Element	# Violations	# Violations/KLOC	# Violations/Methc	Project
com.example.com.example.jacoco	48	1263.2	24.00	com.example.jaco
App.java	24	1263.2	24.00	com.example.jaco
App.java	24	1263.2	24.00	com.example.jaco
LocalVariableNamingConventions	1	52.6	1.00	com.example.jaco
MethodArgumentCouldBeFinal	2	105.3	2.00	com.example.jaco
ShortClassName	1	52.6	1.00	com.example.jaco
CommentRequired	2	105.3	2.00	com.example.jaco
AtLeastOneConstructor	1	52.6	1.00	com.example.jaco
SystemPrintln	1	52.6	1.00	com.example.jaco
LocalVariableCouldBeFinal	1	52.6	1.00	com.example.jaco
OnlyOneReturn	1	52.6	1.00	com.example.jaco
AvoidFinalLocalVariable	1	52.6	1.00	com.example.jaco
PrematureDeclaration	2	105.3	2.00	com.example.jaco
ShortVariable	1	52.6	1.00	com.example.jaco
DataflowAnomalyAnalysis	7	368.4	7.00	com.example.jaco
UnusedLocalVariable	3	157.9	3.00	com.example.jaco
com.example.com.example.jacoco	64	N/A	N/A	com.example.jaco

Violations Outline:



Violations Outline

Pri	Line	created	Rule	Error Message
▶	11	Tue Oct 04 19:48:45 MS	LocalVariableNamingConventions	LocalVariableNamingConventions: The final local variable name 'Size' doesn't match '[a-z][a-zA-Z0-9]*'
▶	15	Tue Oct 04 19:48:45 MS	SystemPrintln	SystemPrintln: System.out.println is used
▶	11	Tue Oct 04 19:48:45 MS	PrematureDeclaration	PrematureDeclaration: Avoid declaring a variable if it is unreferenced before a possible exit point.
▶	5	Tue Oct 04 19:48:45 MS	MethodArgumentCouldBeFinal	MethodArgumentCouldBeFinal: Parameter 'data' is not assigned and could be declared final
▶	9	Tue Oct 04 19:48:45 MS	ShortVariable	ShortVariable: Avoid variables with short names like i
▶	3	Tue Oct 04 19:48:45 MS	CommentRequired	CommentRequired: Class comments are required
▶	18	Tue Oct 04 19:48:45 MS	OnlyOneReturn	OnlyOneReturn: A method should have only one exit point, and that should be the last statement in the method
▶	8	Tue Oct 04 19:48:45 MS	UnusedLocalVariable	UnusedLocalVariable: Avoid unused local variables such as 'tempString'.
▶	5	Tue Oct 04 19:48:45 MS	MethodArgumentCouldBeFinal	MethodArgumentCouldBeFinal: Parameter 'target' is not assigned and could be declared final
▶	8	Tue Oct 04 19:48:45 MS	AvoidFinalLocalVariable	AvoidFinalLocalVariable: Avoid using final local variables, turn them into fields
▶	11	Tue Oct 04 19:48:45 MS	UnusedLocalVariable	UnusedLocalVariable: Avoid unused local variables such as 'Size'.
▶	5	Tue Oct 04 19:48:45 MS	CommentRequired	CommentRequired: Public method and constructor comments are required
▶	3	Tue Oct 04 19:48:45 MS	AtLeastOneConstructor	AtLeastOneConstructor: Each class should declare at least one constructor
▶	8	Tue Oct 04 19:48:45 MS	PrematureDeclaration	PrematureDeclaration: Avoid declaring a variable if it is unreferenced before a possible exit point.
▶	16	Tue Oct 04 19:48:45 MS	LocalVariableCouldBeFinal	LocalVariableCouldBeFinal: Local variable 'mid' could be declared final
▶	10	Tue Oct 04 19:48:45 MS	UnusedLocalVariable	UnusedLocalVariable: Avoid unused local variables such as 'counter'.
▶	3	Tue Oct 04 19:48:45 MS	ShortClassName	ShortClassName: Avoid short class names like App

- **<u>Your evaluation of the tool's usefulness</u>**
- When correctly included into a production pipeline, PMD is a great tool that serves as a quality checker and gate. It makes the source code as efficient as feasible and aids in enforcing coding standards. It does not, however, replace the requirement for testing or code reviews and is not a one-stop solution to testing. Some PMD regulations conflict with one another or even with other rules. Due to its open-source nature, this demonstrates that the tool still has a long, robust development road ahead of it and has a ton of potential.
- The tool can support multiple languages and has out of the box rules for all the languages
- It can easily be integrated into popular build tools and IDEs. Moreover, it's configurable to the extent that you can enable/disable certain functionality as one desires.
- Allows the programmer to write custom rules if required.
- PMD has excellent user documentation. The errors, warnings and APIs are well explained in detail. It also mentioned how these errors are triggered.

References

[1] https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php

[2] https://www.jacoco.org/jacoco/trunk/doc/mission.html

[3] https://pmd.github.io/#home

[4] https://pmd.github.io/pmd-6.42.0/pmd_rules_java.html

[5] https://datarespons.com/code-quality-assurance-with-pmd/