CSE 565 Assignment 3

Member Name: Gautham Vijayaraj
ASU ID: 1229599464
Date: 10/04/2023

Part 1

Select any tool that provides at least decision code coverage:

Tool Description

Jacoco is an open-source code coverage library for Java distributed under the Eclipse Public License. The library is often used in Java projects to measure the coverage of unit or integration tests. But it's also possible to measure the coverage of any test, using a running application, with the Jacoco agent. **(Installation: Can be added as dependency into the pom.xml)**

It provides metrics for **branch, decision, and statement coverage**. It is suitable for use in Maven and Gradle projects and **generates detailed coverage reports**. It's lightweight and easy to integrate with JUnit tests, making it an excellent choice for obtaining code coverage metrics.

Coverage

As mentioned above, Jacoco provides coverage for:

1. **Branch Coverage**: Measures whether each branch (e.g., the true and false parts of an `if` statement) has been executed.
2. **Decision Coverage**: Ensures that each boolean expression has been evaluated both to true and false.
3. **Line Coverage**: Determines which lines of code have been executed.
4. **Instruction Coverage**: Counts the number of executed bytecode instructions.

Source Listing

```
// QuickSort.java
package org.example;

public class QuickSort {

// Applies the QuickSort algorithm to sort the input array
    public void quickSort(int[] array, int low, int high) {
        if (low < high) {
            int partitionIndex = partition(array, low, high);
            quickSort(array, low, partitionIndex - 1);
            quickSort(array, partitionIndex + 1, high);
        }
    }

// Partitions the array around a pivot element.
    private int partition(int[] array, int low, int high) {
        int pivot = array[high];
```

```
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (array[j] < pivot) {
                i++;
                swap(array, i, j);
            }
        }
        swap(array, i + 1, high);
        return i + 1;
    }
// Swaps two elements in the array.
    private void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String[] args) {
        QuickSort sorter = new QuickSort();
        int[] array = {10, 7, 8, 9, 1, 5};
        sorter.quickSort(array, 0, array.length - 1);
        System.out.println("Sorted array: ");
        for (int num : array) {
            System.out.print(num + " ");
        }
    }
}
```

Test Cases

1. **Initial Test Cases.**

```
import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import org.junit.jupiter.api.Test;

public class QuickSortTest {

    @Test
    public void testQuickSortEmptyArray() {
        QuickSort sorter = new QuickSort();
        int[] array = {};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{}, array);
    }

    @Test
    public void testQuickSortSingleElement() {
        QuickSort sorter = new QuickSort();
        int[] array = {1};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1}, array);
    }

    @Test
```

```
    public void testQuickSortAlreadySorted() {
        QuickSort sorter = new QuickSort();
        int[] array = {1, 2, 3, 4, 5};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5}, array);
    }

    @Test
    public void testQuickSortUnsortedArray() {
        QuickSort sorter = new QuickSort();
        int[] array = {5, 3, 2, 8, 1};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 5, 8}, array);
    }

    @Test
    public void testQuickSortWithDuplicates() {
        QuickSort sorter = new QuickSort();
        int[] array = {4, 4, 2, 2, 8, 8};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{2, 2, 4, 4, 8, 8}, array);
    }
}
```

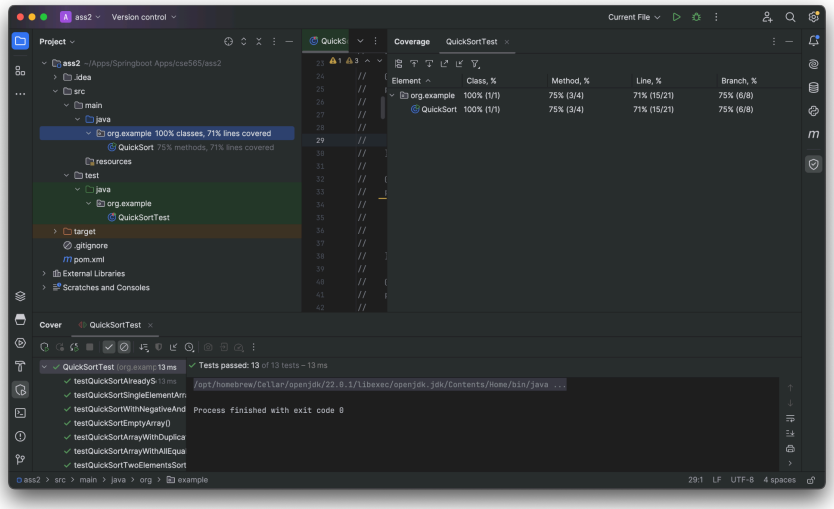
2. Second Set of Test Cases.

```
package org.example;

import static
org.junit.jupiter.api.Assertions.assertArrayEquals;
import org.junit.jupiter.api.Test;

public class QuickSortTest {

    @Test
    public void
testQuickSortWithAllNegativeNumbers() {
        QuickSort sorter = new QuickSort();
        int[] array = {-5, -2, -3, -1, -4};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{-5, -4, -3, -2, -1},
array);
    }

    @Test
    public void
testQuickSortWithMixedPositiveAndNegativeNum
bers() {
        QuickSort sorter = new QuickSort();
        int[] array = {-1, 2, -3, 4, -5};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{-5, -3, -1, 2, 4},
array);
    }

    @Test
    public void testQuickSortWithPivotAtStart() {
        QuickSort sorter = new QuickSort();
        int[] array = {1, 5, 3, 4, 2};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5},
array);
    }

    @Test
    public void testQuickSortWithPivotAtEnd() {
        QuickSort sorter = new QuickSort();
        int[] array = {5, 1, 4, 2, 3};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5},
array);
    }

    @Test
    public void testQuickSortWithPivotInMiddle() {
        QuickSort sorter = new QuickSort();
        int[] array = {3, 1, 2, 5, 4};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5},
array);
    }

    @Test
    public void
testQuickSortWithTwoElementsUnsorted() {
        QuickSort sorter = new QuickSort();
        int[] array = {2, 1};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2}, array);
    }

    @Test
    public void
testQuickSortWithTwoElementsSorted() {
        QuickSort sorter = new QuickSort();
        int[] array = {1, 2};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2}, array);
    }

    @Test
    public void testQuickSortWithEmptyArray() {
        QuickSort sorter = new QuickSort();
        int[] array = {};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{}, array);
```

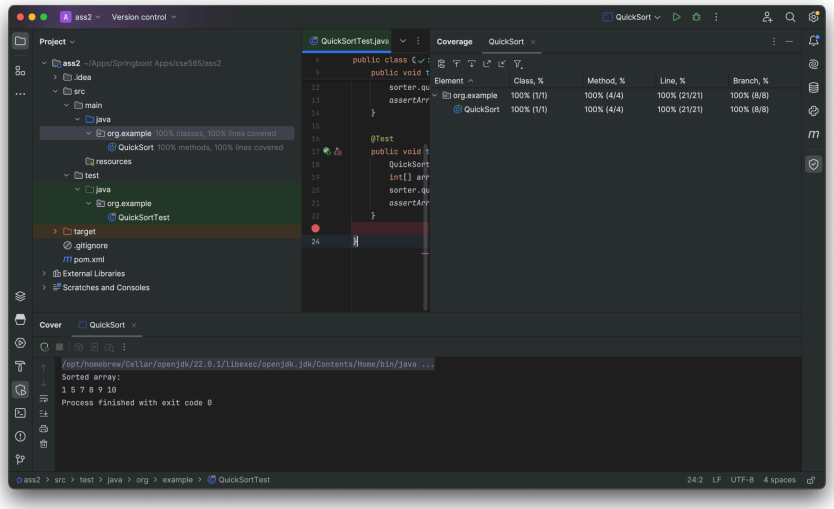
```
    }

    @Test
    public void testQuickSortWithSingleElement() {
        QuickSort sorter = new QuickSort();
        int[] array = {5};
        sorter.quickSort(array, 0, 0);
        assertArrayEquals(new int[]{5}, array);
    }

    @Test
    public void
testQuickSortWithPivotAtBeginningAndEnd() {
        QuickSort sorter = new QuickSort();
        int[] array = {5, 1, 2, 3, 4};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5},
array);
    }

    @Test
    public void
testQuickSortWithDuplicateElements() {
        QuickSort sorter = new QuickSort();
        int[] array = {2, 2, 2, 2, 2};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{2, 2, 2, 2, 2},
array);
    }

    @Test
    public void testQuickSortWithLargeArray() {
        QuickSort sorter = new QuickSort();
        int[] array = new int[100000];
        for (int i = 0; i < array.length; i++) {
            array[i] = (int) (Math.random() * 100000);
        }
        sorter.quickSort(array, 0, array.length - 1);
        for (int i = 1; i < array.length; i++) {
            assert array[i - 1] <= array[i];
        }
    }

    @Test
    public void
testQuickSortWithReverseSortedArray() {
        QuickSort sorter = new QuickSort();
        int[] array = new int[1000];
        for (int i = 0; i < array.length; i++) {
            array[i] = array.length - i;
        }
        sorter.quickSort(array, 0, array.length - 1);
        for (int i = 1; i < array.length; i++) {
            assert array[i - 1] <= array[i];
        }
    }

    @Test
    public void
testQuickSortWithAlreadySortedArray() {
        QuickSort sorter = new QuickSort();
        int[] array = new int[1000];
        for (int i = 0; i < array.length; i++) {
            array[i] = i;
        }
        sorter.quickSort(array, 0, array.length - 1);
        for (int i = 1; i < array.length; i++) {
            assert array[i - 1] <= array[i];
        }
    }

    @Test
    public void
testQuickSortWithAllIdenticalElements() {
        QuickSort sorter = new QuickSort();
        int[] array = {1, 1, 1, 1, 1};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 1, 1, 1, 1},
array);
    }

    @Test
    public void
testQuickSortWithVeryLargeNumbers() {
        QuickSort sorter = new QuickSort();
        int[] array = {1000000, 999999, 888888,
777777, 666666};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{666666, 777777,
888888, 999999, 1000000}, array);
    }

    @Test
    public void testQuickSortWithMinIntValues() {
        QuickSort sorter = new QuickSort();
        int[] array = {Integer.MIN_VALUE, -1, 0, 1,
Integer.MIN_VALUE};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new
int[]{Integer.MIN_VALUE, Integer.MIN_VALUE, -1,
0, 1}, array);
    }

    @Test
    public void testQuickSortWithMaxIntValues() {
        QuickSort sorter = new QuickSort();
        int[] array = {Integer.MAX_VALUE, 100,
Integer.MAX_VALUE - 1, Integer.MAX_VALUE};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{100,
Integer.MAX_VALUE - 1, Integer.MAX_VALUE,
Integer.MAX_VALUE}, array);
    }

    @Test
    public void testQuickSortWithNullArray() {
        QuickSort sorter = new QuickSort();
        int[] array = null;
        try {
            sorter.quickSort(array, 0, array.length - 1);
        } catch (NullPointerException e) {
            System.out.println("Handled
NullPointerException for null array.");
        }
    }

    @Test
    public void
testQuickSortWithPositiveValuesOnly() {
        QuickSort sorter = new QuickSort();
        int[] array = {15, 10, 20, 5, 30};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{5, 10, 15, 20,
30}, array);
    }

    @Test
    public void
testQuickSortWithPartiallySortedArray() {
        QuickSort sorter = new QuickSort();
```

```
        int[] array = {1, 2, 3, 6, 5, 4};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 2, 3, 4, 5, 6},
array);
    }

    @Test
    public void testQuickSortWithGapValues() {
        QuickSort sorter = new QuickSort();
        int[] array = {1000, 1, 500, 250, 750};
        sorter.quickSort(array, 0, array.length - 1);
        assertArrayEquals(new int[]{1, 250, 500, 750,
1000}, array);
    }
```

Screenshots



After First Test Cases



After Second Test Cases

Evaluation

JaCoCo enhanced the quality of my Java projects by providing clear insights into test coverage. It enabled me to **identify untested sections of code** and improve test cases accordingly. This helped **me ensure critical logic paths were covered**, resulting in fewer bugs and more reliable code. JaCoCo's intuitive interface and lightweight performance made it straightforward to use, helping me maintain a high standard of code quality throughout the development process. This modified version of the QuickSort class helps in visually validating all paths and ensures that each scenario is thoroughly tested.

Part 2

Select any tool that performs static source code analysis:

Tool Description

SonarLint is an IDE extension that helps you detect and fix quality issues as you write code. Like a spell checker, SonarLint squiggles flaws so that they can be fixed before committing code.

SonarLint will identify various **code quality issues, bugs, and data flow** anomalies such as **uninitialized variables, dead code, unused variables, reassignments of variables**. It can be integrated into IntelliJ to perform local analysis without requiring a connection to a SonarQube server.

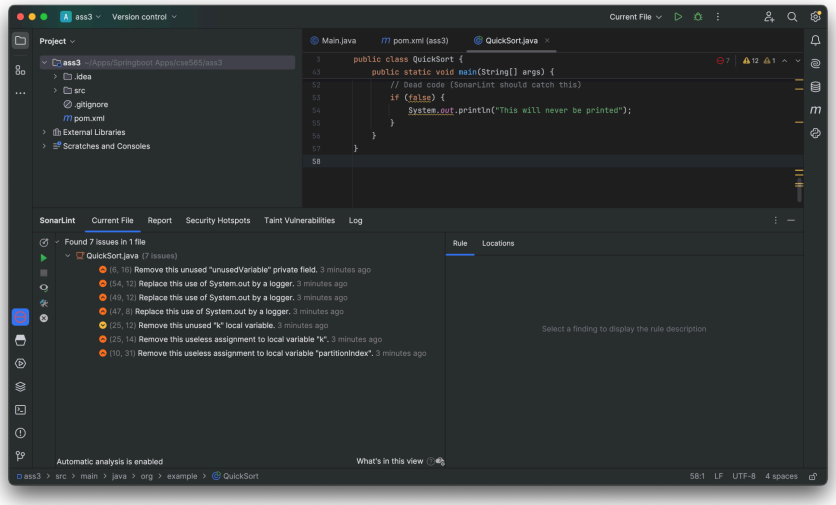
Installation: Installed as a plugin using the IntelliJ IDE.

Source Listing

```
// QuickSort Class from above and I've added some deliberate anomalies for SonalLint to detect
package org.example;

public class QuickSort {

    // Example of an unused variable (SonarLint should catch this)
    private int unusedVariable = 10;

    public void quickSort(int[] array, int low, int high) {
        if (low < high) {
            int partitionIndex = partition(array, low, high);

            // Reassignment of variable 'partitionIndex' without usage
            partitionIndex = 5; // This should be flagged as a data flow anomaly

            quickSort(array, low, partitionIndex - 1);
            quickSort(array, partitionIndex + 1, high);
        }
    }

    private int partition(int[] array, int low, int high) {
        int pivot = array[high];
        int i = low - 1;

        // Unused variable 'k' (SonarLint should catch this)
```

```
        int k = pivot;

        for (int j = low; j < high; j++) {
            if (array[j] < pivot) {
                i++;
                swap(array, i, j);
            }
        }
        swap(array, i + 1, high);
        return i + 1;
    }

    private void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String[] args) {
        QuickSort sorter = new QuickSort();
        int[] array = {10, 7, 8, 9, 1, 5};
        sorter.quickSort(array, 0, array.length - 1);
        System.out.println("Sorted array: ");
        for (int num : array) {
            System.out.print(num + " ");
        }

        // Dead code (SonarLint should catch this)
        if (false) {
            System.out.println("This will never be printed");
        }
    }
}
```

Screenshot



Analysis performed by SonarLint

Evaluation

SonarLint proved to be **highly effective in identifying data flow anomalies** such as unused variables and reassignments without usage. It **highlighted code smells and potential bugs** that are not immediately apparent during code review. However, it might produce **false positives for some scenarios** and **cannot evaluate logical errors** specific to business requirements.

Conclusion

JaCoCo is a code coverage analysis tool that provides insights into which parts of the code are executed when running test cases. It is primarily used to measure the effectiveness of the test suite by showing which lines, branches, and methods are covered. JaCoCo helps identify gaps in testing by providing detailed metrics such as class, method, and line coverage, making it an essential tool for ensuring that critical parts of the code are adequately tested. It focuses on runtime analysis and requires test case execution to generate coverage reports, but it does not identify static issues in the code that may lead to potential bugs.

SonarLint, on the other hand, is a static code analysis tool that analyzes code without executing it. It is used to detect code quality issues such as unused variables, dead code, and potential bugs related to data flow. SonarLint works in real-time as you code, highlighting issues and offering suggestions for code improvement. Unlike JaCoCo, SonarLint does not require test case execution, making it highly effective for identifying coding errors, code smells, and refactoring opportunities. However, it does not provide information on how well your test cases cover the code, as it does not measure runtime coverage.

Both tools complement each other well, with JaCoCo focusing on dynamic analysis of test coverage and SonarLint providing static analysis to maintain code quality. Together, they offer a comprehensive solution for ensuring both the quality and coverage of your code.

References

ChatGPT Prompt: https://chatgpt.com/share/6700b42b-8134-8013-9cfd-af877f08689f
ChatGPT Prompt 2: https://chatgpt.com/share/6700baa7-265c-8013-bea3-7f8fb26dc1b4
JaCoCo Github Repository: GitHub - jacoco/jacoco: :microscope: Java Code Coverage Library
JaCoCo Installation Steps: JaCoCo Java Code Coverage Library
SonarLint Documentation: SonarLint - IntelliJ IDEs Plugin | Marketplace
Gemini Chat Prompt: https://g.co/gemini/share/ff7619b14594

Note: There is one more link to another ChatGPT prompt which I'm unable to generate because links won't be generated for conversations which contain user uploaded images. I had uploaded the screenshot of my ide output because my code was failing. I can send screenshots of that prompt if that is required.