CSE 565 Assignment 1

Member Name: Gautham Vijayaraj
ASU ID: 1229599464
Date: 09/6/2023

Part 1

1. **Ask ChatGPT to describe how Generative AI can be used to generate unit test cases.**
   I. Automatic Code Analysis: To analyze the source code to identify the key components, functions, and methods that require testing.
   II. Test Case Creation Based on Input/Output: To generate test cases by learning from the function's input-output behavior.
   III. Generating Edge and Corner Cases: To create test cases that include edge cases, such as boundary values, negative inputs, or unexpected data types.
   IV. Test Coverage Enhancement: To analyze existing test cases and compare them with the actual code to identify areas that lack sufficient test coverage.
   V. Learning from Test Patterns: By learning from historical test case data or commonly used testing frameworks, Generative AI can generate new unit tests that follow best practices.
   VI. Refining and Optimizing Tests: To suggest improvements or optimizations for existing unit tests, such as reducing redundancy, improving test efficiency, or fixing brittle test cases that may fail due to minor changes in the code.
   VII. Natural Language to Code Conversion: To convert natural language descriptions of what a test should verify into actual test code.
   VIII. Continuous Integration and Regression Testing: To automatically generate unit tests as the code evolves, ensuring that new changes do not break existing functionality.
2. **Identify a tool that can implement the approach briefly describing its capabilities.**
   I. Tool: Diffblue Cover

   Diffblue Cover is an AI-powered tool that automatically generates unit test cases for Java applications based on the input-output behavior of functions. It uses machine learning algorithms to analyze code and create tests that reflect real-world scenarios and edge cases. The tool is designed to integrate with CI/CD pipelines, continuously generating tests as the codebase evolves.

II. Capabilities:

i. **Automated Unit Test Generation:** It generates unit tests by examining code behavior, covering normal, edge, and exceptional cases.

ii. **Test Coverage Analysis:** Diffblue Cover ensures that generated tests help increase overall test coverage.

iii. **Seamless Integration:** Works with popular build tools like Maven and Gradle, making it easy to integrate into existing development environments.

iv. **Refactoring Awareness:** The tool adapts to code refactoring and updates tests to match the new code structure.

III. Implementation:

i. Download and Install Diffblue Cover

ii. Add Diffblue Cover to your project

iii. Select Diffblue Cover > Create Tests

iv. Diffblue Cover will automatically create unit tests based on your project

3. Compare the tool with Junit identifying how they differ and how they might work together.

I. Comparison with Junit:

i. **Test Creation:** Diffblue Cover generates test cases automatically, while JUnit relies on developers manually writing test cases.

ii. **AI vs. Manual:** Diffblue Cover uses AI to analyze code behavior and generate test cases, while JUnit gives developers full control over the logic of the tests.

iii. **Coverage Focus:** Diffblue Cover is more focused on test coverage optimization, whereas JUnit requires developers to decide the scope of the tests.

II. How they may work together:

i. **Complementary Use:** Diffblue Cover can be used to automatically generate baseline tests that ensure adequate coverage of basic functionality and edge cases. JUnit can then be used by developers to manually write more customized and specific tests that reflect business logic or more complex scenarios.

ii. **Boosting Productivity:** Diffblue Cover can handle the more repetitive, coverage-focused tests, while JUnit can allow for more focused, developer-driven testing. This combination can save time and increase productivity, allowing developers to focus on higher-value test creation.

iii. **Improving Test Quality**: Automated tests generated by Diffblue Cover might lack some domain-specific assertions. Developers can modify and enhance these tests using JUnit, fine-tuning them for specific business requirements.

Part 2

1. **Write code to implement the Quick Sort algorithm.**

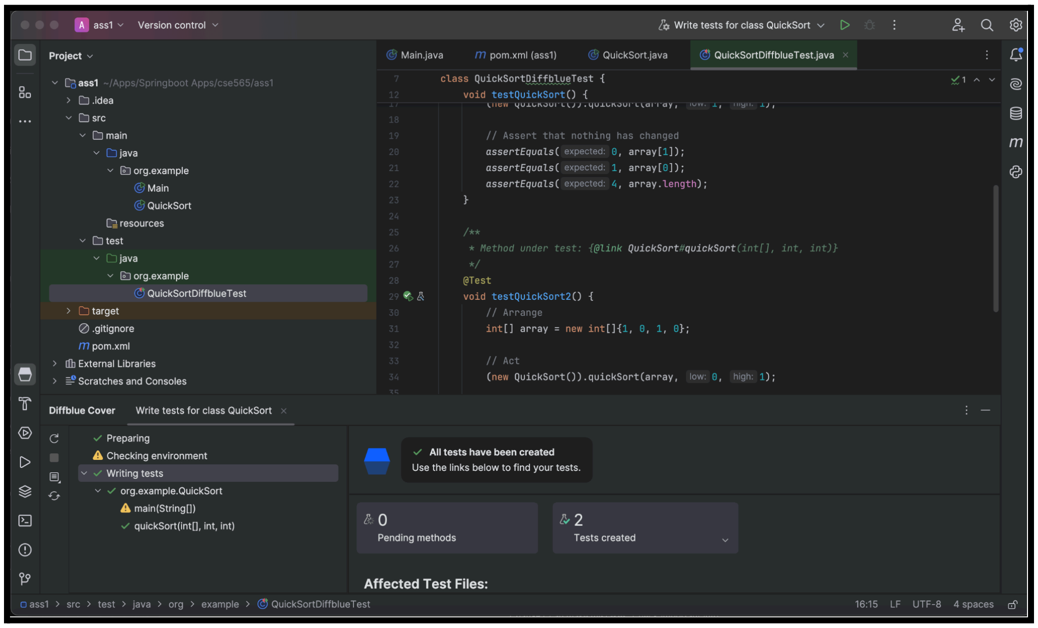
```
public class QuickSort {

    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);
        }
    }

    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;
    }

    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};
        int n = array.length;
        sorter.quickSort(array, 0, n - 1);
        System.out.println("Sorted array: ");
        for (int num : array) {
            System.out.print(num + " ");
        }
    }
}
```

2. **Utilize a unit testing framework of your choice to test your code. Submit a copy of the code to test.**

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class QuickSortDiffblueTest {
    /**
     * Method under test: {@link QuickSort#quickSort(int[], int, int)}
     */
    @Test
    void testQuickSort() {
        // Arrange
        int[] array = new int[]{1, 0, 1, 0};

        // Act
        (new QuickSort()).quickSort(array, 1, 1);

        // Assert that nothing has changed
        assertEquals(0, array[1]);
        assertEquals(1, array[0]);
        assertEquals(4, array.length);
    }

    /**
     * Method under test: {@link QuickSort#quickSort(int[], int, int)}
     */
    @Test
    void testQuickSort2() {
        // Arrange
        int[] array = new int[]{1, 0, 1, 0};

        // Act
        (new QuickSort()).quickSort(array, 0, 1);

        // Assert
        assertEquals(0, array[0]);
        assertEquals(1, array[1]);
        assertEquals(4, array.length);
    }
}
```

3. Submit a copy of the output report.

TEST CASES GENERATED USING DIFFBLUE COVER



```java
class QuickSortDiffblueTest {
    void testQuickSort() {

        // Assert that nothing has changed
        assertEquals( expected: 0, array[1]);
        assertEquals( expected: 1, array[0]);
        assertEquals( expected: 4, array.length);
    }

    /**
     * Method under test: {@link QuickSort#quickSort(int[], int, int)}
     */
    @Test
    void testQuickSort2() {
        // Arrange
        int[] array = new int[]{1, 0, 1, 0};

        // Act
        (new QuickSort()).quickSort(array,  low: 0,  high: 1);
```

Diffblue Cover — Write tests for class QuickSort

- ✓ Preparing
- ⚠ Checking environment
- ✓ Writing tests
  - ✓ org.example.QuickSort
    - ⚠ main(String[])
    - ✓ quickSort(int[], int, int)

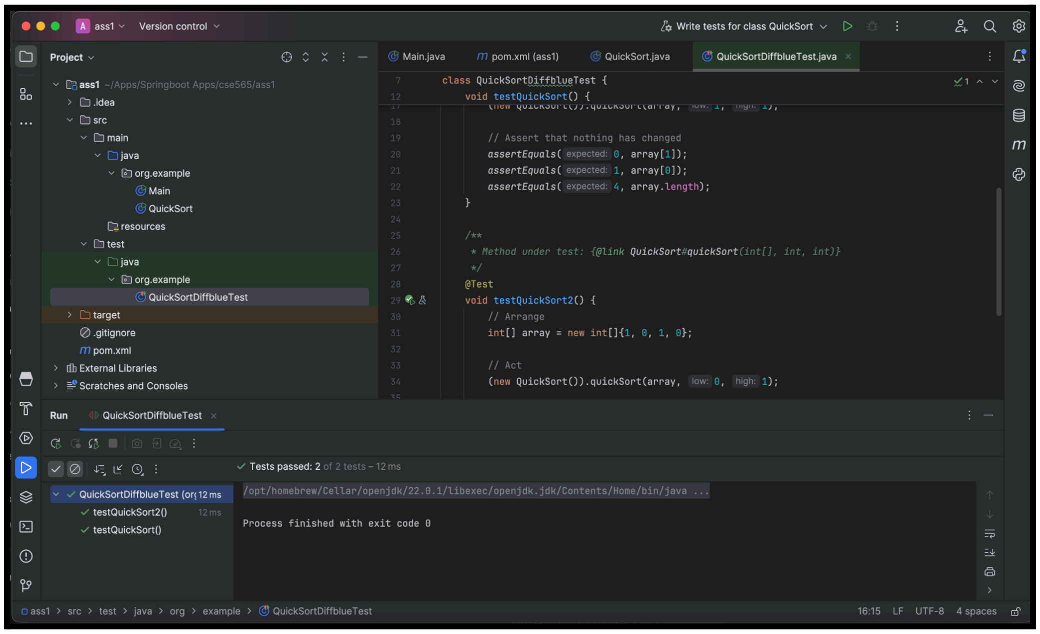
✓ **All tests have been created**
Use the links below to find your tests.

0 Pending methods

2 Tests created

Affected Test Files:

BOTH THE TEST CASES PASSED: 2 OUT OF 2



Run — QuickSortDiffblueTest

✓ Tests passed: 2 of 2 tests – 12 ms

- ✓ QuickSortDiffblueTest (or; 12 ms
  - ✓ testQuickSort2() 12 ms
  - ✓ testQuickSort()

```
/opt/homebrew/Cellar/openjdk/22.0.1/libexec/openjdk.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

ass1
Version control
Write tests for class QuickSort
Main.java
pom.xml (ass1)
QuickSort.java
QuickSortDiffblueTest.java
Project
ass1 > src > test > java > org > example > QuickSortDiffblueTest
16:15 LF UTF-8 4 spaces

References:

https://chatgpt.com/share/59fdc4f2-1582-431e-aa9c-b48e02b4ce72 - ChatGPT Responses

https://docs.diffblue.com/get-started/get-started/get-started-cover-plugin - Diffblue Cover Documentation