Student Information

First Name: Yazan

Last Name: Mousa

Student number: 500823471

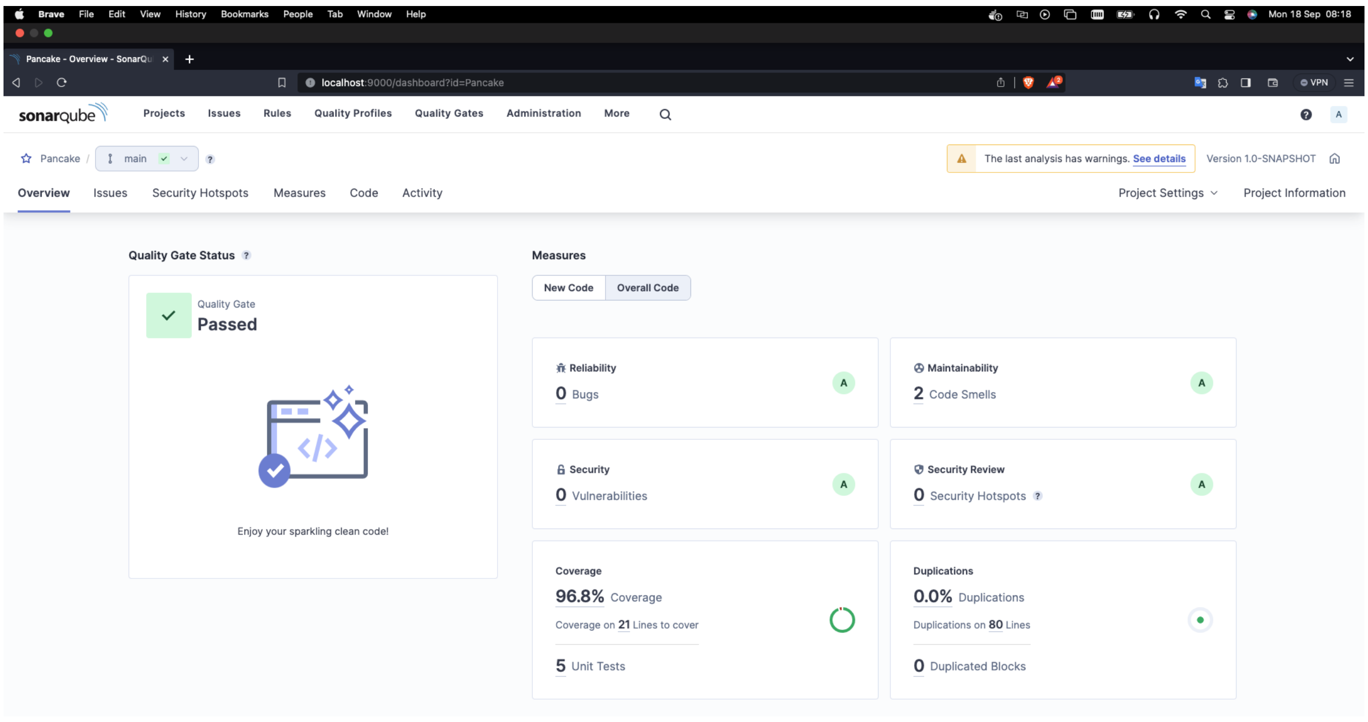
Assignment

1. Git log

```
Place here the results of the following command: git log --
pretty=format:"%hx %ad%x09%s" --date=short`

973679bx 2023-09-18     updated readme
fc50bbcx 2023-09-18     improved sortingclass with feedback from classmate
e4bfe8fx 2023-09-18     Improved test with feedback from classmate
2ef9a67x 2023-09-18     added screenshots for readme
05d08e1x 2023-09-15     Updated Readme
7c6eacfx 2023-09-15     Sonarqube screenshots added
b0fac43x 2023-09-15     create readme.md in the right path
fc18484x 2023-09-15     Delete old readme
f403bcfx 2023-09-15     Plugin added to pom.xml for Sonarqube to run
449e27fx 2023-09-14     Pancake sorting algorithm passes the
testSortReverseOrder test.
107976bx 2023-09-14     Pancake sorting algorithm passes the
testSortSinglePancake test.
f350c41x 2023-09-14     Pancake sorting algorithm passes the
testSortEmptyArray test.
dc26f5dx 2023-09-14     Test created.
73c1935x 2023-09-14     Create starter project
0f296a4x 2023-09-14     Initial commit
```

2. Sonarqube

A dated screenshot of the overview of the following quality gates(https://docs.sonarqube.org/latest/user-guide/quality-gates/): Reliability, Security,Maintainability, Coverage and Duplications. Provide a short discussion of the results.


Brave File Edit View History Bookmarks People Tab Window Help
Mon 18 Sep 08:18
Pancake - Overview - SonarQu
localhost:9000/dashboard?id=Pancake
sonarqube
Projects Issues Rules Quality Profiles Quality Gates Administration More
Pancake / main
Version 1.0-SNAPSHOT
Overview Issues Security Hotspots Measures Code Activity
Project Settings Project Information
New Code Overall Code

The last analysis has warnings. See details

Quality Gate Status

Quality Gate
Passed

Enjoy your sparkling clean code!

Measures

Reliability
0 Bugs A

Maintainability
2 Code Smells A

Security
0 Vulnerabilities A

Security Review
0 Security Hotspots A

Coverage
96.8% Coverage
Coverage on 21 Lines to cover
5 Unit Tests

Duplications
0.0% Duplications
Duplications on 80 Lines
0 Duplicated Blocks

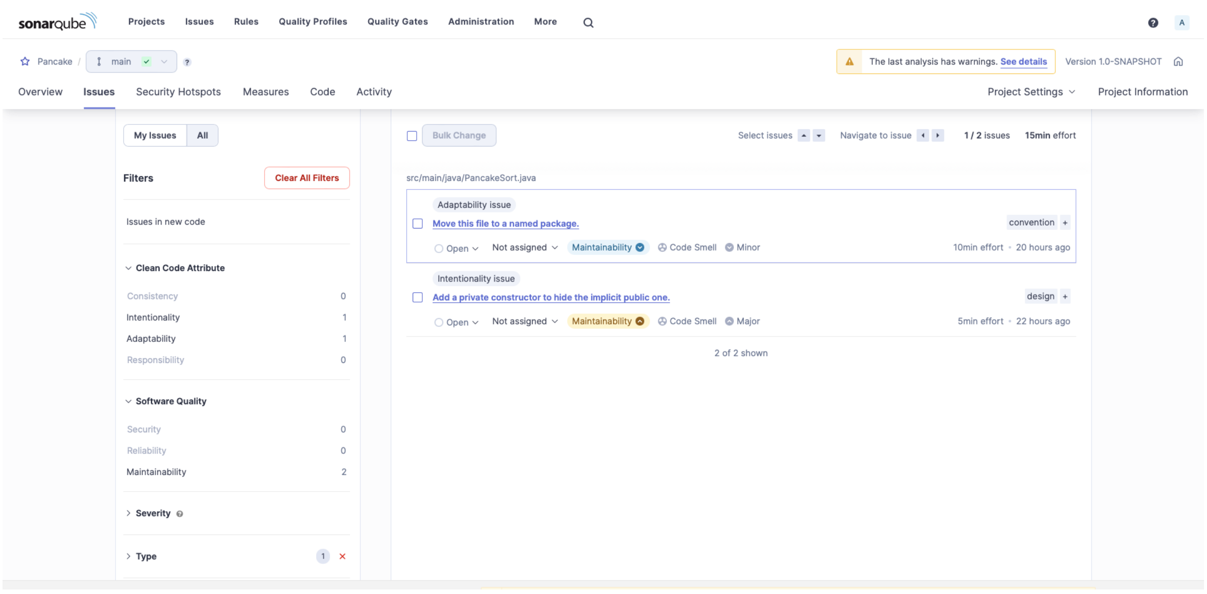
Project Metrics

Reliability

- **Score:** A
- **Bugs:** 0
  - The number of bugs is 0, indicating that there are no identified issues in this category. Additionally, the evaluation rates the project's handling of bugs as "A", signifying a high level of reliability.

Maintainability

- **Score:** A
- **Code Smells:** 2
  - The maintainability score is 2, suggesting that there are some areas that may require attention to improve maintainability. However, the presence of code smells is rated as "A", indicating that there are no significant issues in this regard.
  - The two code smells were not crucial for this project, because it only contains one class and one test file.
  - The first one was about moving the Pancake class into a package, but there are no different models/ class.
  - The second one was about creating a private constructor, but Pancake class is not initialized via its constructor in any file.

- 

The last analysis has warnings. See details
Version 1.0-SNAPSHOT

Select issues Navigate to issue **1 / 2** issues **15min** effort

Filters

Issues in new code

Clean Code Attribute

Consistency	0
Intentionality	1
Adaptability	1
Responsibility	0

Software Quality

Security	0
Reliability	0
Maintainability	2

Severity

Type 1

src/main/java/PancakeSort.java

Adaptability issue
Move this file to a named package. — convention
Open · Not assigned · Maintainability · Code Smell · Minor — 10min effort · 20 hours ago

Intentionality issue
Add a private constructor to hide the implicit public one. — design
Open · Not assigned · Maintainability · Code Smell · Major — 5min effort · 22 hours ago

2 of 2 shown

Security

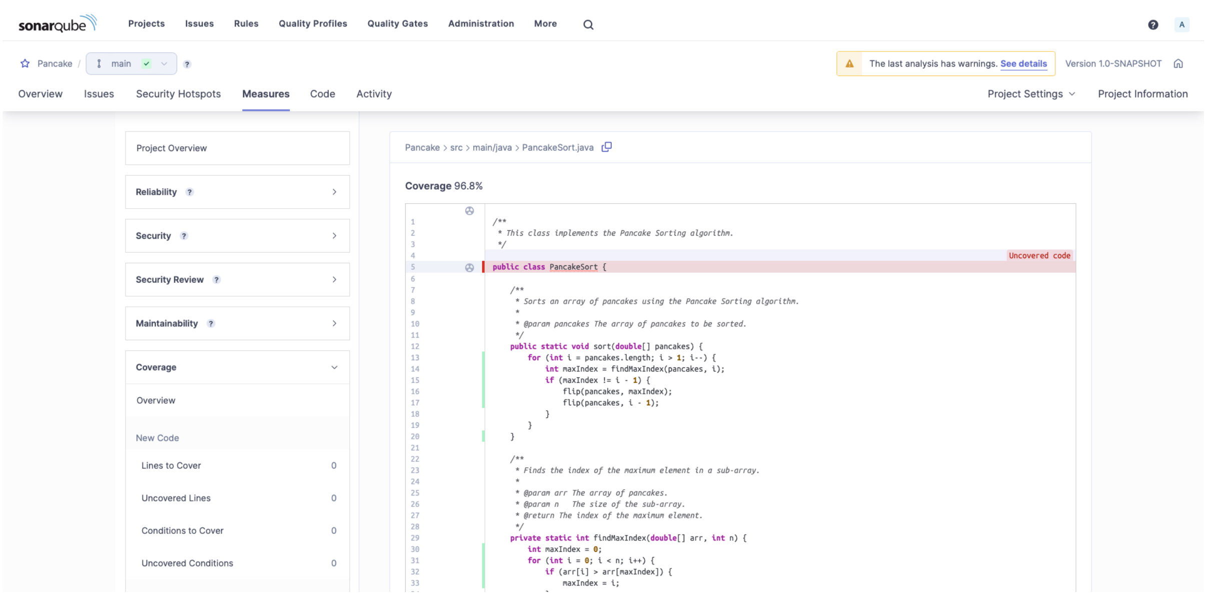
- **Score:** A
- **Vulnerabilities:** 0
    - The number of vulnerabilities is 0, indicating that there are no identified security vulnerabilities in the project. Furthermore, the assessment rates the project's security as "A", indicating a high level of security.

Security Review

- **Score:** A
- **Security Hotspots:** 0
    - The number of security hotspots is 0, suggesting that there are no identified security hotspots in the project. This indicates a robust security posture.

Coverage

- **Percentage:** 96.8%
    - The coverage percentage is 96.8%, indicating that a significant portion of the codebase is covered by tests. There is one portion of the code that should still be covered. The name of the Pancake class is not covered in the test. There is basically only one Pancake class in the project.

- 

Coverage 96.8%

```java
/**
 * This class implements the Pancake Sorting algorithm.
 */

public class PancakeSort {

    /**
     * Sorts an array of pancakes using the Pancake Sorting algorithm.
     *
     * @param pancakes The array of pancakes to be sorted.
     */
    public static void sort(double[] pancakes) {
        for (int i = pancakes.length; i > 1; i--) {
            int maxIndex = findMaxIndex(pancakes, i);
            if (maxIndex != i - 1) {
                flip(pancakes, maxIndex);
                flip(pancakes, i - 1);
            }
        }
    }

    /**
     * Finds the index of the maximum element in a sub-array.
     *
     * @param arr The array of pancakes.
     * @param n   The size of the sub-array.
     * @return The index of the maximum element.
     */
    private static int findMaxIndex(double[] arr, int n) {
        int maxIndex = 0;
        for (int i = 0; i < n; i++) {
            if (arr[i] > arr[maxIndex]) {
                maxIndex = i;
```

Duplications

- **Percentage:** 0.0%
- **Lines of unique code:** 80
    - The duplication percentage is 0.0%, indicating that there are no identified code duplications. There are 80 lines of code that are considered unique.

3. Test Driven Development

Your best test class code snippets with a rationale why the unit tests are “good” tests. Provide a link to the Test class and the class under test in Git.

The unit tests for the Pancake Sorting algorithm have been designed with the following considerations in mind:

1. **Comprehensive Coverage**:

- The unit tests cover a wide range of scenarios, including edge cases and boundary conditions. This ensures that the algorithm is thoroughly evaluated under various input scenarios.

2. **Scenario Validation**:

- The tests include scenarios such as empty arrays, arrays with a single element, and arrays with different orderings. This helps validate the correctness of the algorithm in diverse situations.

3. **Clarity and Readability**:

- Test method names are clear and descriptive, providing a clear indication of the scenario being evaluated. This makes it easy for developers to understand the purpose of each test.

4. **Expected Results**:

- Each test includes an array of expected results, serving as a reference for the correct output after the sorting operation. This facilitates straightforward verification of algorithm accuracy.

5. **Precision Handling**:

- The use of a small precision tolerance (0.0001) in the `assertArrayEquals` statements demonstrates consideration for potential floating-point precision issues, ensuring results are within an acceptable margin of error.

6. **Error Handling Evaluation**:

- Additional test cases have been included to assess error handling capabilities. These cover scenarios where the input violates specified constraints, such as arrays with more than 25 pancakes or arrays with negative values.

7. **Test Independence**:

- Each test is independent, operating in isolation without reliance on the state or outcome of other tests. This promotes robustness and allows for individual test execution.

8. **Minimal External Dependencies**:

- Tests do not rely on external dependencies or require complex setup. This ensures that they can be executed in any environment without additional configuration.

9. **Test Readability and Formatting**:

- Tests are well-organized and formatted, enhancing readability for developers. This aids in quick comprehension of the testing logic.

By adhering to these principles, the unit tests contribute to a comprehensive testing suite that thoroughly evaluates the correctness and reliability of the Pancake Sorting algorithm. They serve as a valuable tool for ensuring code quality and robustness.

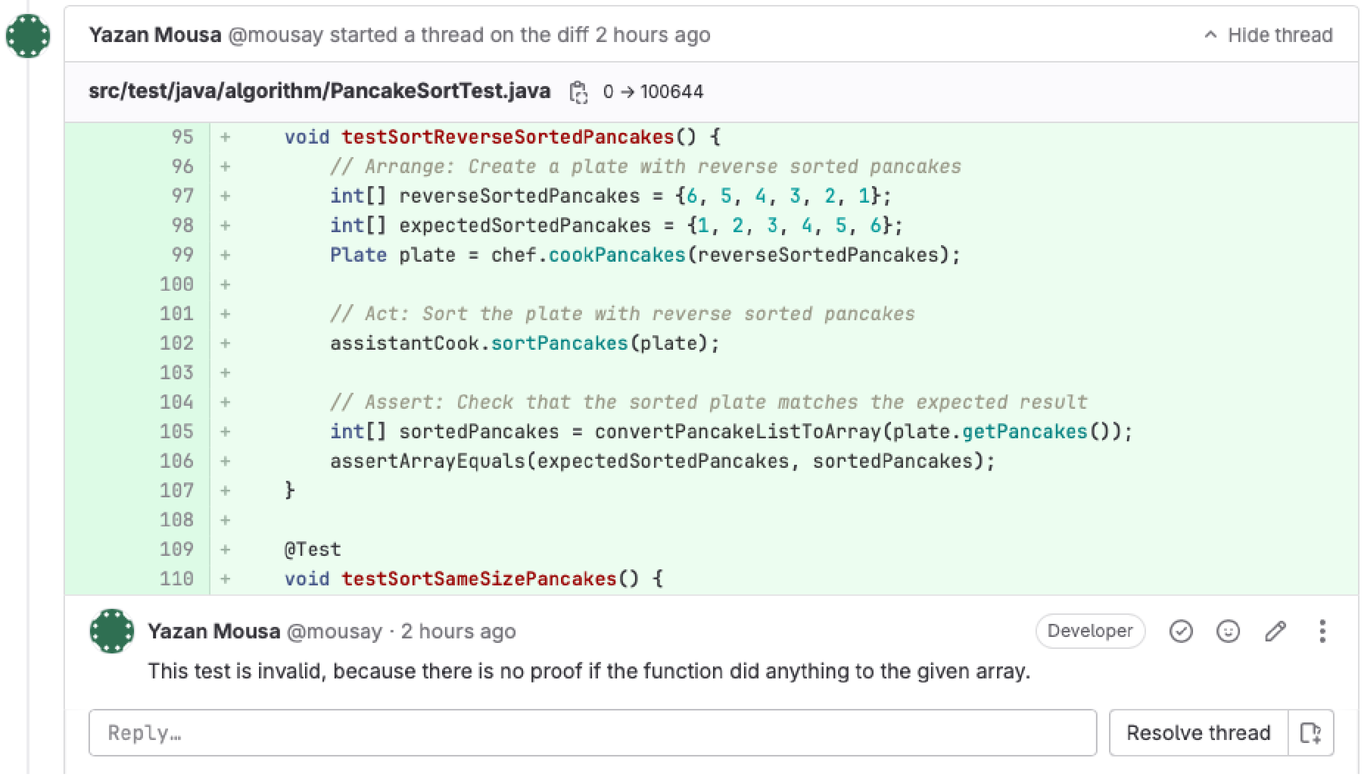
```
// Testing sorting an empty array
@Test
public void testSortEmptyArray() {
        double[] pancakes = {};
        double[] expected = {};

        // Sorting the pancakes
        PancakeSort.sort(pancakes);

        // Asserting that the sorted array matches the expected array
        assertArrayEquals(expected, pancakes, 0.0001);
        }

// Testing sorting a single pancake
@Test
public void testSortSinglePancake() {
        double[] pancakes = {5.5};
        double[] expected = {5.5};

        // Sorting the pancakes
        PancakeSort.sort(pancakes);

        // Asserting that the sorted array matches the expected array
        assertArrayEquals(expected, pancakes, 0.0001);
        }

// Testing sorting an already sorted array
@Test
public void testSortAlreadySorted() {
        double[] pancakes = {1.0, 2.5, 3.0, 4.2, 5.7};
        double[] expected = {1.0, 2.5, 3.0, 4.2, 5.7};

        // Sorting the pancakes
        PancakeSort.sort(pancakes);

        // Asserting that the sorted array matches the expected array
        assertArrayEquals(expected, pancakes, 0.0001);
        }

// Testing sorting a reversed order array
@Test
public void testSortReverseOrder() {
        double[] pancakes = {5.7, 4.2, 3.0, 2.5, 1.0};
        double[] expected = {1.0, 2.5, 3.0, 4.2, 5.7};

        // Sorting the pancakes
        PancakeSort.sort(pancakes);

        // Asserting that the sorted array matches the expected array
        assertArrayEquals(expected, pancakes, 0.0001);
        }

// Testing sorting an array with mixed order
@Test
public void testSortMixedOrder() {
        double[] pancakes = {3.2, 1.5, 4.0, 5.7, 2.3};
        double[] expected = {1.5, 2.3, 3.2, 4.0, 5.7};

        // Sorting the pancakes
        PancakeSort.sort(pancakes);

        // Asserting that the sorted array matches the expected array
        assertArrayEquals(expected, pancakes, 0.0001);
        }

// Testing sorting an array with non-numeric values
@Test
public void testSortNonNumericValues() {
        double[] pancakes = {3.2, 1.5, 4.0, Double.NaN, 2.3}; // Contains
a non-numeric value (NaN)
        assertThrows(IllegalArgumentException.class, () ->
PancakeSort.sort(pancakes));
        }

// Testing sorting an array with more than 25 pancakes
@Test
public void testSortMoreThanMaxPancakes() {
        double[] pancakes = new double[26]; // Creating an array with 26
pancakes
        assertThrows(IllegalArgumentException.class, () ->
PancakeSort.sort(pancakes));
        }

// Testing sorting an array with negative pancake values
@Test
public void testSortNegativePancakes() {
        double[] pancakes = {-3.0, -1.5, -4.0, -5.7, -2.3};
        assertThrows(IllegalArgumentException.class, () ->
PancakeSort.sort(pancakes));
        }
```

link-to-your-class-under-test

link-to-your-test-class

4. Code Reviews

Screenshots of the code reviews you have performed on code of another student as comments in Gitlab: Provide a link to the comments in Gitlab.



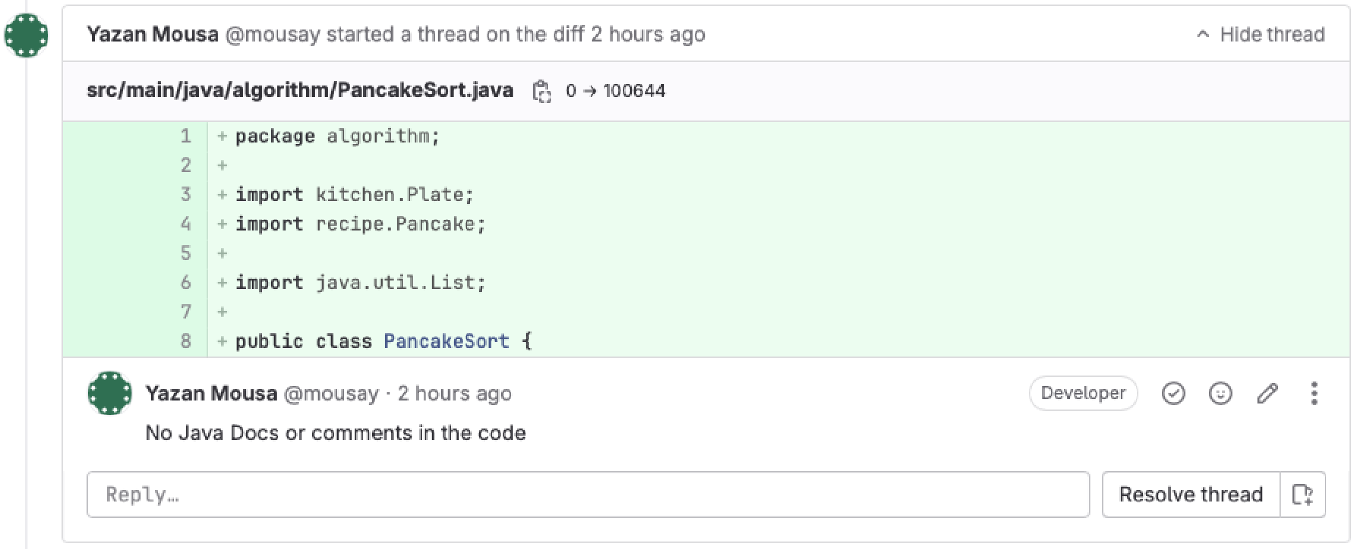
Yazan Mousa @mousay started a thread on the diff 2 hours ago

src/test/java/algorithm/PancakeSortTest.java 0 → 100644

```
+    void testSortReverseSortedPancakes() {
+        // Arrange: Create a plate with reverse sorted pancakes
+        int[] reverseSortedPancakes = {6, 5, 4, 3, 2, 1};
+        int[] expectedSortedPancakes = {1, 2, 3, 4, 5, 6};
+        Plate plate = chef.cookPancakes(reverseSortedPancakes);
+
+        // Act: Sort the plate with reverse sorted pancakes
+        assistantCook.sortPancakes(plate);
+
+        // Assert: Check that the sorted plate matches the expected result
+        int[] sortedPancakes = convertPancakeListToArray(plate.getPancakes());
+        assertArrayEquals(expectedSortedPancakes, sortedPancakes);
+    }
+
+    @Test
+    void testSortSameSizePancakes() {
```

Yazan Mousa @mousay · 2 hours ago

This test is invalid, because there is no proof if the function did anything to the given array.



Yazan Mousa @mousay started a thread on the diff 2 hours ago

Hide thread

src/main/java/algorithm/PancakeSort.java 0 → 100644

```
+ package algorithm;
+
+ import kitchen.Plate;
+ import recipe.Pancake;
+
+ import java.util.List;
+
+ public class PancakeSort {
```

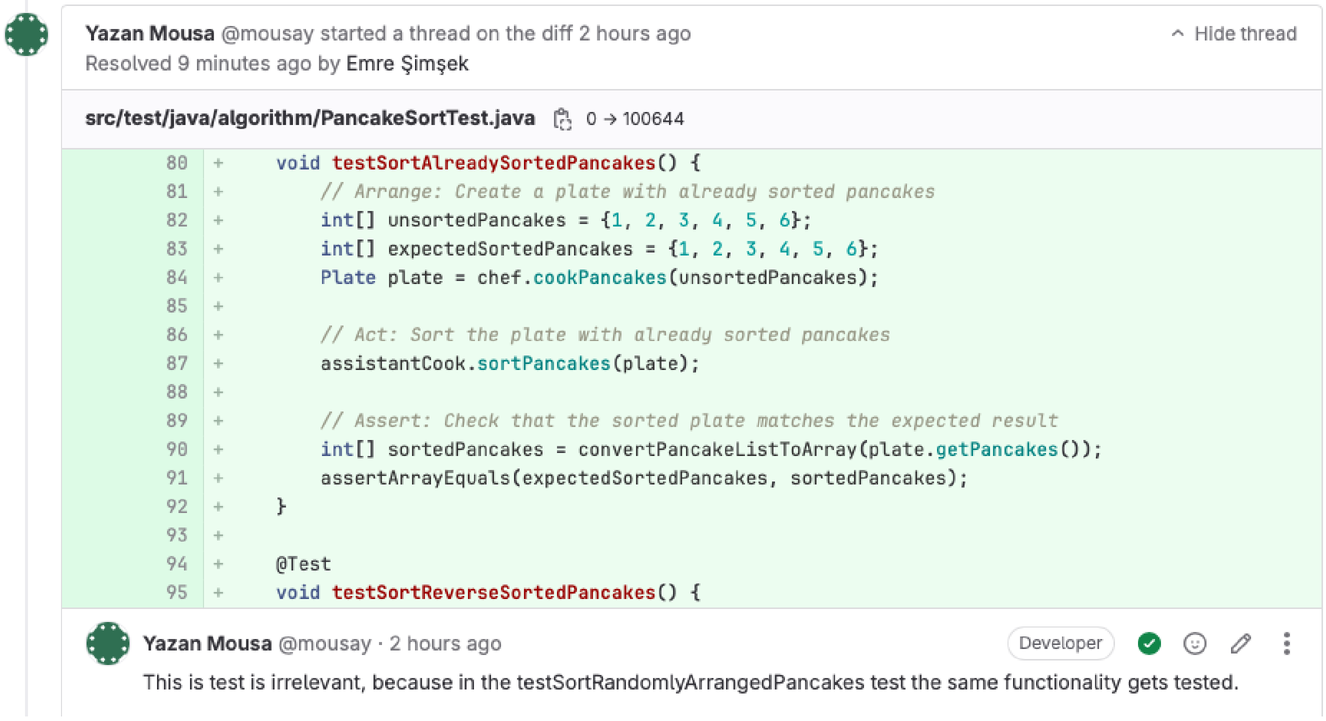
Yazan Mousa @mousay · 2 hours ago

Developer

No Java Docs or comments in the code

Reply…

Resolve thread



Yazan Mousa @mousay started a thread on the diff 2 hours ago
Resolved 9 minutes ago by Emre Şimşek

Hide thread

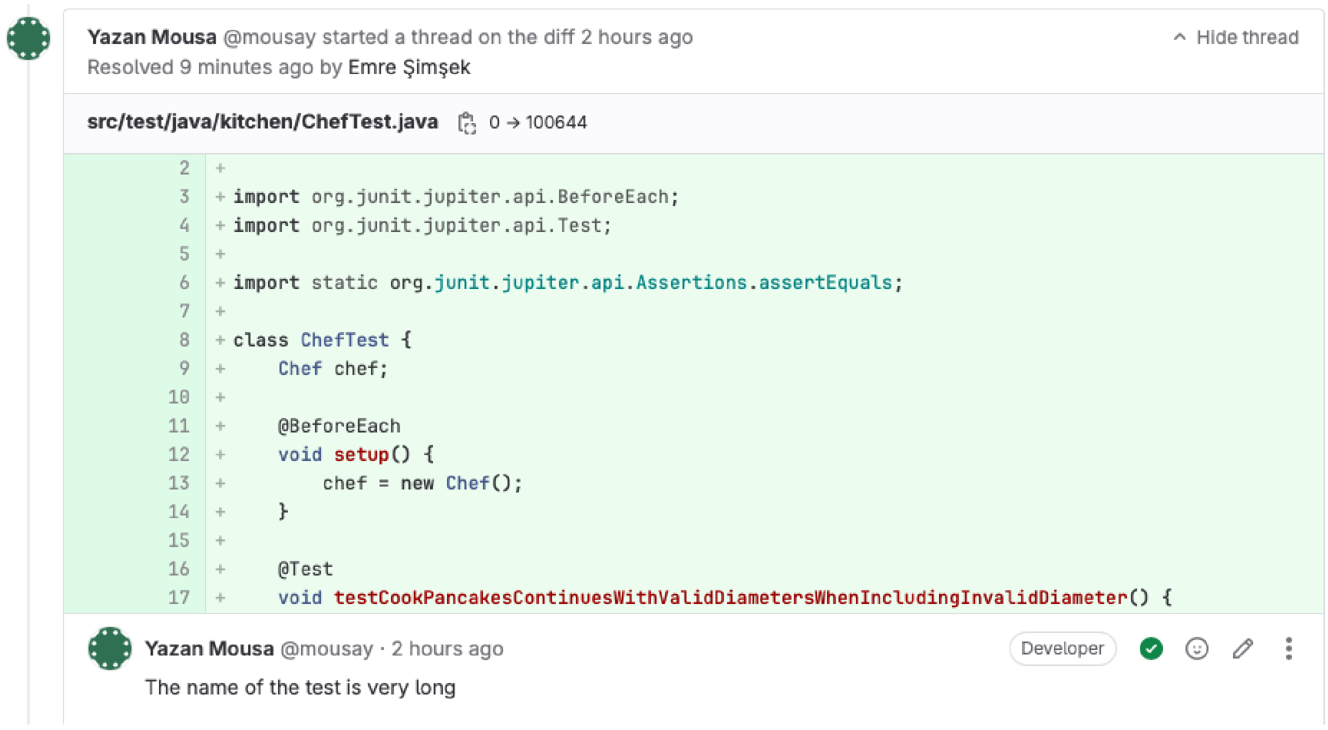
src/test/java/algorithm/PancakeSortTest.java 0 → 100644

```
+     void testSortAlreadySortedPancakes() {
+         // Arrange: Create a plate with already sorted pancakes
+         int[] unsortedPancakes = {1, 2, 3, 4, 5, 6};
+         int[] expectedSortedPancakes = {1, 2, 3, 4, 5, 6};
+         Plate plate = chef.cookPancakes(unsortedPancakes);
+
+         // Act: Sort the plate with already sorted pancakes
+         assistantCook.sortPancakes(plate);
+
+         // Assert: Check that the sorted plate matches the expected result
+         int[] sortedPancakes = convertPancakeListToArray(plate.getPancakes());
+         assertArrayEquals(expectedSortedPancakes, sortedPancakes);
+     }
+
+     @Test
+     void testSortReverseSortedPancakes() {
```

Yazan Mousa @mousay · 2 hours ago

Developer

This is test is irrelevant, because in the testSortRandomlyArrangedPancakes test the same functionality gets tested.



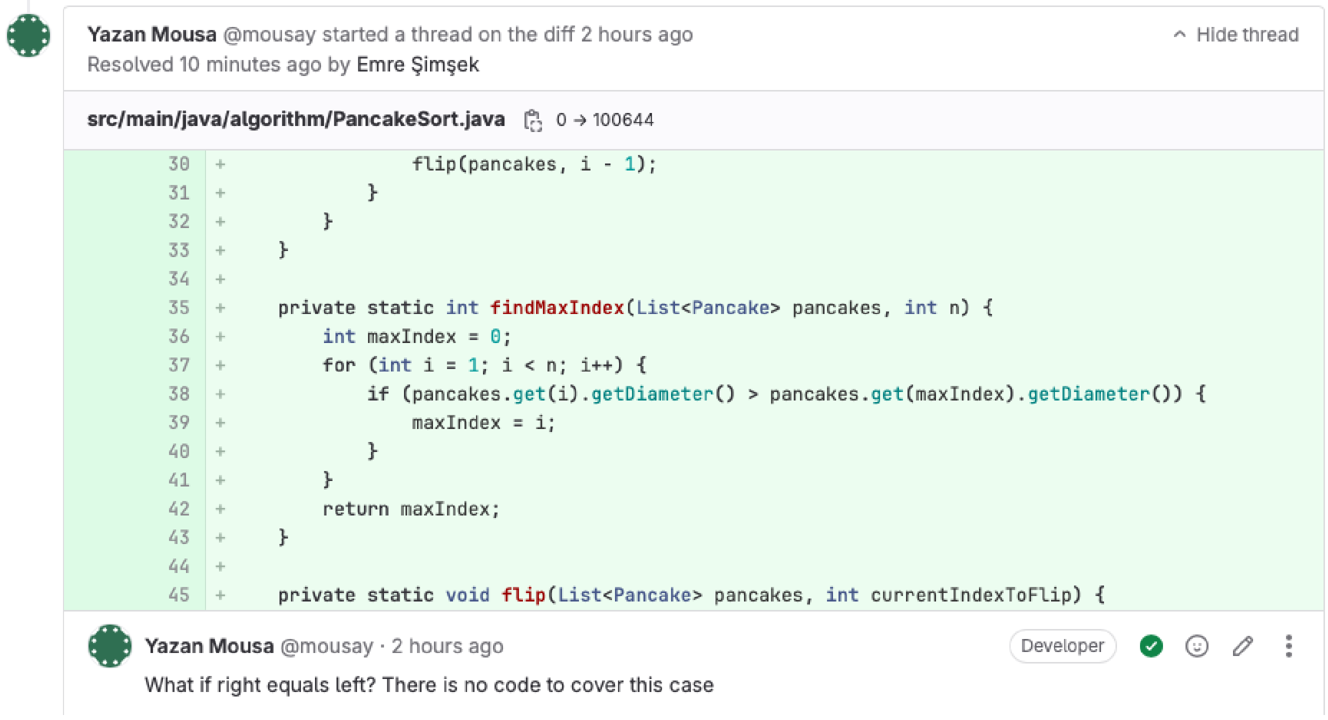
Yazan Mousa @mousay started a thread on the diff 2 hours ago
Resolved 9 minutes ago by Emre Şimşek

src/test/java/kitchen/ChefTest.java 0 → 100644

```
+
+ import org.junit.jupiter.api.BeforeEach;
+ import org.junit.jupiter.api.Test;
+
+ import static org.junit.jupiter.api.Assertions.assertEquals;
+
+ class ChefTest {
+     Chef chef;
+
+     @BeforeEach
+     void setup() {
+         chef = new Chef();
+     }
+
+     @Test
+     void testCookPancakesContinuesWithValidDiametersWhenIncludingInvalidDiameter() {
```

Yazan Mousa @mousay · 2 hours ago Developer

The name of the test is very long



Yazan Mousa @mousay started a thread on the diff 2 hours ago
Resolved 10 minutes ago by Emre Şimşek

src/main/java/algorithm/PancakeSort.java 0 → 100644

```
+                 flip(pancakes, i - 1);
+             }
+         }
+     }
+
+     private static int findMaxIndex(List<Pancake> pancakes, int n) {
+         int maxIndex = 0;
+         for (int i = 1; i < n; i++) {
+             if (pancakes.get(i).getDiameter() > pancakes.get(maxIndex).getDiameter()) {
+                 maxIndex = i;
+             }
+         }
+         return maxIndex;
+     }
+
+     private static void flip(List<Pancake> pancakes, int currentIndexToFlip) {
```

Yazan Mousa @mousay · 2 hours ago Developer

What if right equals left? There is no code to cover this case

link-to-comments-you-gave