**9914b Test Plan Review**

**Section 1**

The purpose and scope of the test plan are well written, and the system overview has a good explanation of the procedure to be followed in the test plan. The overview of the document describes the contents and purpose of each section concisely.

**Section 2**

Test section provide a good model of the class and the methods to be tested. However, providing a brief description of the methods in the class could help on specifying what aspects of the methods are being evaluated.

**Section 3**

The format is clean and describes the objectives of the test in detail, as well as their priority. However, there are several tests evaluating the Table class, and few test evaluating only the method isSorted() of the TableSorter class. Consider using tests 3-5 to evaluate the method sortable() in the TableSorter class.

**Section 4**

The following section reviews the overall structure and grammar of the tests in section 4 of test plan 9914, and provides the result of the tests.

**Test 1**

Steps are clear and easy to follow due to good description of the purpose and expected results. Additionally, the comments clarify doubts in step 2. However, the description of the expected result in step 3 could be improved by mentioning to use the appendix section as reference; this can ensure that the expected result was successfully reached.

The following test result was produced by implementing test 1:

Checking the table can be generated for a square number of inputs

Input: 9 8

6 7

Result: The table with a square number of inputs is created

* [[I@d706f19]

9 8

7 6

**Test 2**

In addition to comments, a good description of operator actions could clear doubts in the test procedure. As in test 1, the expected result in step 3 could refer to appendix to ensure the result has been achieved. Nonetheless, steps are easy to follow due to the good description of the purpose and expected results.

The following test result was produced by implementing test 2:

Checking the table will not be generated for a non-square number of inputs

Input: 9 8 7

6 5 4

Result: The table was not created, and an Exception error was issued by Junit and the console.

Function: In this test, the Exception errors prove that the table will not be created if the number of inputs is not square.

* java.lang.Exception (by Junit)
* ERROR!: Input must be square. (By Console)

**Test 3**

There is a noticeable change in the naming convention of the test text file in the purpose of step 2, making the implementation of the tests slightly inconsistent; to maintain organized and clean tests, it is recommended to rename the test text files in tests 1 and 2 following the format used in test 3. Also, the improvements mentioned in test 2 can be applied in this test. Nevertheless, the test is easy to follow and implement.

The following test result was produced by implementing test 3:

Checking the table can be generated with positive integers

Input: 5 7

9 8

Result: The table was created with positive integers

Function: In this test, the print statement and the output in the console proves that the table will be created if the input has positive integers.

* [[I@7ee955a8]

5 7

9 8

**Test 4**

There is a small spelling error in the purpose of the second and third step where ‘x-4’ should be corrected to -4. Also, the improvements mentioned in test 2 can be applied in this test. However, the test is easy to follow and implement.

The following test result was produced by implementing test 4:

Checking the table can be generated with negative integers

Input: -9 -8

-4 -3

Result: The table was created with negative integers

* System.***out***.println(t4.toString());
* [[I@7ee955a8]

-9 -8

-4 -3

**Test 5**

The steps are easy to follow, and the test case is easy to implement. However, the tests until now evaluate the class Table; which is not bad since it ensures that the class works properly and can be used by others, but consider using test 1-5 on TableSorter to ensure that isSorted() and sorted() methods work too. Also, the improvements mentioned in test 2 can be applied in this test.

The following test result was produced by implementing test 5:

Checking the table can be generated with two positive integers, zero, and a negative integer

Input: 5 1

0 -3

Result: The table was created with two positive integers, a zero, and a negative integer

* [[I@48fa0f47]

5 1

1. -3

**Test 6**

The steps are concise and clear. However, the objective has an ambiguous description on the method being tested in class TableSorter, which in this case is isSorted(). Also, the improvements mentioned in test 2 can be applied in this test.

The following test result was produced by implementing test 6:

Checking the method isSorted() returns true for a sorted table

Input: -1 0

2 3

Result: The method isSorted() returned true

**Test 7**

The steps are easy to follow, and the test case is easy to implement. There is a small spelling error in the purpose of the second step where ‘0. 7, -5 …’, should be changed to ‘0, 7, -5 …’. Also, the improvements mentioned in test 2 can be applied in this test.

The following test result was produced by implementing test 7:

Checking the method isSorted() returns false for an unsorted table

Input: 1 9 2

8 0 7

-5 4 6

Result: The method isSorted() returned false and failed the test.

**Section 5**

No comments.

**Section 6**

The section might not be necessary. Consider moving the software requirements to Section 1.

**Section 7**

The figures provide a good example for the expected outputs and the structure of the directory, I would recommend mentioning the appendix in Section 4 to ensure that the tester follows the procedures correctly.

**Summary**

* Table is evaluated from tests 1-5
* TableSorter is only evaluated on its method isSorted() in tests 6-7
* TableSorter method sortable() was not tested
* No significant errors were found following the test plan; only the expected error for the creation of an nxm table