SortTable Black Box Testing

Test plan

Version 1.0

02/12/2020

**Document Control**

**Approval**

The Guidance Team shall approve this document.

**Document Change Control**

|  |  |
| --- | --- |
| Initial Release: | 1.0 |
| Current Release: | 1.0 |
| Indicator of Last Page in Document: | $ |
| Date of Last Review: | 2/12/2020 |
| Date of Next Review: | TBD |
| Target Date for Next Update: | TBD |

**Distribution List**

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Guidance Team Members: Dr. Steve Roach

Software Team Members:

**Change Summary**

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 02/12/2020 |  | Consolidated entire document. |
|  |  |  |  |
|  |  |  |  |

Note: The template presented in this document was taken from:

Donaldson, S., and S. Siegel, *Successful Software Development*. Upper Saddle River, NJ: Prentice Hall, 2001, pp. 321-323.

Note: The template presented in this document was taken from: Donaldson, S., and S. Siegel, *Successful Software Development*. Upper Saddle River, NJ: Prentice Hall, 2001, pp. 321-323 and modified by Humberto Mendoza and Steve Roach.

Supplementary information is from:

Pfleeger, S. *Software Engineering, Theory and Practice*. Upper Saddle River, NJ: Prentice Hall, 1998, p. 365.

**Table of Contents**

[**DOCUMENT CONTROL II**](#_heading=h.30j0zll)

[**Approval ii**](#_heading=h.2xcytpi)

[**Document Change Control ii**](#_heading=h.1ci93xb)

[**Distribution List ii**](#_heading=h.3whwml4)

[**Change Summary ii**](#_heading=h.2bn6wsx)

[**1.**](#_heading=h.3dy6vkm) **INTRODUCTION 1**

**1.1.** **Purpose 1**

**1.2.** **Scope 1**

**1.3.** **System Overview 1**

**1.4.** **Suspension and Exit Criteria 1**

**1.5.** **Document Overview 1**

**1.6.** **References 1**

[**2.**](#_heading=h.qsh70q) **TEST ITEMS AND FEATURES 2**

[**3.**](#_heading=h.35nkun2) **TESTING APPROACH 3**

[**4.**](#_heading=h.44sinio) **TEST XX 4**

**4.1.** **Test <<test id>> 4**

[**5.**](#_heading=h.3as4poj) **USER INTERFACE TESTING 5**

[**6.**](#_heading=h.3j2qqm3) **TEST SCHEDULE 6**

[**7.**](#_heading=h.1y810tw) **OTHER SECTIONS 7**

[**8.**](#_heading=h.1pxezwc) **APPENDIX 8**

# Introduction

This project aims to test the robustness and detect bugs of the java class SortTable. The elements to be tested are 2 classes: ‘isSorted’ and ‘sortTable’. The testing approach is black box testing, specifically boundary testing.

## Purpose

The overall strategy for the testing is as follows: set up the testing environment, feed sets of inputs from the test cases, and compare the findings with the expected output. The project testing is entirely based on black box testing, this means the internal workings of the class are unknown. Due to the nature of the testing and time constraints the testing will focus on boundary testing. A finite number of tests will be provided to the tester along with a set of instructions on how to perform said testing. Lastly, valid and invalid test cases will be provided to assess the functionality and exception handling of the TableSorter class.

## Scope

N/A

## System Overview

N/A

## Suspension and Exit Criteria

The exit criteria is as follows: 80% of the normal/non-critical test cases must pass. This is because some of the test cases include tests that are not essential for the functionality of the class. Also, all critical tests must pass. Suspension criteria: none; all test cases will be executed, due to the amount of test provided.

## Document Overview

N/A

## References

N/A

# Test Items and Features

**2.1. Test items:**

The test items are part of the java class TableSorter, isSorted and SortTable. Both are instance methods with the following specifications:

* **IsSorted:** The method ‘isSorted’ returns true if every row and column of an a given table (object).
  + Method signature: *public boolean isSorted(Table t)*
* **SortTable**: The method ‘sortTable’ sorts a Table so that isSorted() is true.
  + Method signature: *public static void sortable (Table t)*

**2.2. Features:**

Table class:

The table class is provided as an external feature of the testing. This is because the methods to be tested utilize objects of type ‘Table’ which the table class provides.

See appendix at the end of the document for details and information about the class.

TableSorter class:

The tableSorter class is provided when conducting testing, however these tests were made without the prior knowledge of the contents of the class.

Runner class:

The runner class is provided to initialize the corresponding objects, namely the table to be analyzed from the text.txt file. See appendix for the entire file.

Test.txt:

The test.txt file is used to provide a table to be constructed to the runner class.

# Testing Approach

The testing approach involves the tester using the runner class provided on the appendix. This class initializes the corresponding variables, and prints the table to be analyzed (which comes from the test.txt file), and whether or not it is sorted. Then it calls the sortTable method and prints the output and the results of isSorted after performing the output. The tester only has to manually modify the test.txt file in order to run the tests and record the actual output. Note: the test cases are simple enough for the tester to determine if in fact a table is sorted or not based on the previously stated specifications. The runner class and test.txt files have to reside within the package cs5387. A modern Integrated Development Environment (IDE) is not required but it is prefered. The tester must unflatten the test case inputs when writing them into the test.txt file. For example, an input of [1,2,3,4] should look like this on the test.txt file:

1 2

3 4

This requires the number of columns to be the same as the number of rows. Based on the specification of making a NxN table.

# Test Cases

**4.1 Test sets:**

The test sets contain sets of inputs to feed into the test.txt file.

**4.1.1 IsSorted Test Case** input**:**

1. (empty table of size 0, the file is completely blank) : true
2. [0] (table with a single element) : true
3. [-1] (table with negative element) :true
4. [Integer.MAX\_VALUE] (table with single biggest element) : true
5. [Integer.MAX\_VALUE + 1] (table with values outside of domain) : false; error
6. [1,2,3,4] (sorted table with normal set of values 2x2) : true
7. [4,3,2,1] (unsorted table with normal set of values 2x2) :false

**4.1.2 sortTable:**

1. (empty table of size 0, the file is completely blank)
2. [0] (table with a single element)
3. [-1] (table with negative element)
4. [Integer.MAX\_VALUE] (table with single biggest element)
5. [Integer.MAX\_VALUE + 1] (table with values outside of domain) possible error
6. [1,2,3,4] (sorted table with normal set of values 2x2)
7. [4,3,2,1] (unsorted table with normal set of values 2x2)

# User Interface Testing

To handle runtime errors and compile-time errors refer to the bullets below:

* Runtime possible errors:
  + Invalid input, table contains non-numeric characters:
    - Possible solution: Revise text input characters within the ‘text.txt’ file are all numeric values from ‘0’ to ’9’.
    - Possible solution: Revise text input characters within the ‘text.txt’ file are all and are properly separated by a space and each row ends with a new line.
* Compile-time errors:
  + Invlaid input (type matching):
    - Possible solution: Make sure the passed parameters to SortTable and IsSorted are correctly assigned as their signature requests them.

For example, tring to pass a ‘Double’ to an ‘Integer’ type.

# Test Schedule

All tests and recordings are to be done in a single session, time expected of the session: less than one hour (after setting up the testing environment).

# Other Sections

N/A

# Appendix

**Runner class:**

//Beginning of class

package cs5387;

import java.io.FileNotFoundException;

import java.io.IOException;

public class Runner {

public static void main(String[] args) throws FileNotFoundException, IOException, Exception {

Table t1 = null;

t1 = Table.GetTable("test.txt");

printTable(t1);

TableSorter mach= new TableSorter();

System.out.println(mach.isSorted(t1));

TableSorter.sortTable(t1);

printTable(t1);

System.out.println(mach.isSorted(t1));

}

public static void printTable(Table t ){

for (int i=0; i<t.getSize(); ++i) {

for (int j=0; j<t.getSize(); ++j) {

System.out.print(t.getTableValue(i, j) + " ");

}

System.out.println();

}

}

}

//End of class

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Test.txt:**

The file test.txt is a simple file consisting of the table to be used during testing. Some rules apply to the structure of how it has to be formatted. Below is a good example of how valid input should look like:

8 1

2 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table class:**

Class Table holds an NxN table of integers. The Table interface has 4 methods:

1. public static Table getTable (String filename) returns an instance of table initialized with values from the file filename.
2. public int getTableValue(row, column) returns the integer at (row, column)
3. public void setTableValue(int row, int column, int value) sets the value of the table at row, column to value getSize() returns N

$