**Black-Box Test Plan:**

***METHOD 1 TEST PLAN:***

*public boolean isSorted(Table t) returns true if every row and every column of the table t is sorted in ascending order*

Being that the input of this method is an object Table, we need to test what happens when you input something other than a Table. For this we shall use an equivalence class where it ensures that the input being received is of type “Table.”

***Equivalence Classes (Partitioning):***

|  |  |  |
| --- | --- | --- |
| **Input** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| An NxN array such that:  Table t | Table t = {1,2,3,4,  5,6,7,8,  9,1,2,3}  Table t = {1,2,  3,4,} | int t = 0  int[] t = {1,2,3,4}  int[][] t = {1,2,3,4,5,6,7,8,9,...} |

Being that the required output is to be a boolean, we need to test all the possible outcomes of it coming out as true or false. There are only two possible outputs, so that means there isn’t an apparent equivalence class or boundary value analysis that can be created for this aspect. The approach for this is to use a decision table. The reason why we should test this is to make sure that there aren't any false positives when giving out the result of this check.

***Decision Table Testing:***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sorted** | True | True | True | True | False | False | False | False |
| **Not Sorted** | False | False | True | True | False | False | True | True |
| **Partially Sorted** | True | False | True | False | True | False | True | False |
|  |  |  |  |  |  |  |  |  |
| **It is sorted** |  | Checkmark |  |  |  |  |  |  |
| **It is NOT sorted** | Checkmark |  |  |  | Checkmark |  | Checkmark | Checkmark |
| **Error/Invalid (not possible)** |  |  | Checkmark | Checkmark |  | Checkmark |  |  |

***METHOD 2 TEST PLAN:***

*public static void sortable (Table t) sorts a Table so that isSorted() is true*

For the next required method, the instructions were to sort the table. The output is a Table, so this means that we need to test that the method does give the output of the table. Since we already made the tests for “isSorted”, the part about checking if it is true moves to those tests.

The first approach for this is to also create an equivalence class. Knowing that it also takes in a Table as an input, the same test as above also applies for this method.

***Equivalence Classes (Partitioning):***

|  |  |  |
| --- | --- | --- |
| **Input** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| An NxN array such that:  Table t | Table t = {1,2,3,4,  5,6,7,8,  9,1,2,3}  Table t = {1,2,  3,4,} | int t = 0  int[] t = {1,2,3,4}  int[][] t = {1,2,3,4,5,6,7,8,9,...} |

Now we need to check that the method did not modify the data itself, only rearranged them. In other words, we need to check that the table that was sorted was not somehow changed into a completely different table. This could be in terms of size, data, etc. So the way to approach this is also with a decision table.

***Decision Table Testing:***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Same Table** | True | True | True | True | False | False | False | False |
| **Modified Contents** | False | False | True | True | False | False | True | True |
| **Changed in size** | True | False | True | False | True | False | True | False |
|  |  |  |  |  |  |  |  |  |
| **Table kept the same (sorted or not)** |  | Checkmark |  |  |  |  |  |  |
| **Table and its contents have been modified** |  |  |  | Checkmark |  |  | Checkmark | Checkmark |
| **Invalid** | Checkmark |  | Checkmark |  | Checkmark | Checkmark |  |  |

***OTHER - TEST PLAN:***

For both of the methods required, it was also known that the table needed to be a square. For this aspect, we need an equivalence class in order to test the possible sizes of tables this method receives. This can also involve boundary value analysis. However, since this involves the class Table.java, these tests might have already been done. This section is where we took these inputs into consideration and planned to test them if necessary. This essentially needs to be tested before one can continue with “isSorted” since it takes in what Table.java generated.

Here are the possible equivalence classes that should be considered when creating the table to pass on to the isSorted methods.

***Equivalence Classes (Partitioning):***

|  |  |  |
| --- | --- | --- |
| **Input** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| A Table t such that:  0 >= t.size >= 2,147,483,647 | t.size = 1  t.size = 4  t.size = 100  t.size = 5 | t.size = 2,147,483,648  t.size = -1 |
| A Table t such that:  t.size = N  = k  and  N =  (table is square) | t.size = 4  t.size = 100  t.size = 1000 | t.size = 5  t.size = 7  t.size = 90  t.size = 1 |

One thing to take note of for this equivalence classes is that, although the Table size might be a valid input for a regular table, the method requires for the table to be a square. This means that some of the valid inputs for the first one could be considered invalid for the second one because it does not meet the required NxN square.

***Boundary Value Analysis:***

|  |  |
| --- | --- |
| **Input** | **Boundary Cases** |
| A Table t such that:  0 >= t.size >= 2,147,483,647 | -1, 0, 2147483647, 2147483648 |

For this particular type of testing, the part about checking if it is a square doesn’t have an exact value to measure boundaries, which is why it wasn’t added on to this. Most likely, all of these equivalence classes and boundary value analysis have to be tested all in different combinations due to them relating heavily with each other.