**南京工程学院**

**上机实 验 报 告**

课 程 名 称： 软件质量保证与测试

实验项目名称： 黑盒测试

实验成绩评定：

# 2021-2022-2学期

# Task 1

In this task, you will assume that you are testing an e-commerce system. The exercise is to create functional tests for the web page that accepts the orders.  A screen prototype for the order-entry web page shown in Fig 1. The system accepts a five-digit numeric item ID number from 00000 to 99999. In the product catalog in the system database, these item Ids are sorted by price, with the cheapest items having the lower (closest to 00000) item Id numbers and the most expensive items having the higher (closest to 99999) item Id numbers. You do not have to worry about testing the sorting of data in the database, though, as you are not testing the data-entry process for the catalog. The system accepts a quantity to be ordered, from 1 to 99. If the user enters a previously ordered item ID and a 0 quantity to be ordered, that item is removed from the shopping cart. Based on these inputs, the system retrieves the item price, calculates the item total (quantity times item price), and adds the item total to the cart total. Due to limits on credit card orders that can be processed, the maximum cart total is $999.99.

# 1.1 Testing item ID and Quantity

Table 1 is the condition table for item ID and quantity.

Table 1. Condition table for item ID and Quantity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Class type** | **Class** | **Boundary** | **Tag** | **TC** |
| **Item Id** | valid | [0-99999] | 100 | ID \_Normal | T7-T12,T13 |
| valid | [0-99999] | 0 | ID \_V1 | T1 |
| valid | [0-99999] | 1 | ID \_V2 | T2 |
| valid | [0-99999] | 99998 | ID \_V3 | T3 |
| valid | [0-99999] | 99999 | ID \_V4 | T4 |
| valid | [0-99999] | 5000 | ID \_V5 | T14 |
| valid | [0-99999] | 5001 | ID \_V6 | T15 |
| invalid | ItemID <0 | -1 | ID \_IV1 | T5 |
| invalid | ItemID >99999 | 100000 | ID \_IV2 | T6 |
| **Quantity** | valid | [1-99] | 10 | Q\_Normal | T1-T6,T13 |
| valid | [1-99] | 1 | Q\_V1 | T7 |
| valid | [1-99] | 2 | Q\_V2 | T8 |
| valid | [1-99] | 98 | Q\_V3 | T9 |
| valid | [1-99] | 99 | Q\_V4 | T10 |
| valid | [1-99] | 40 | Q\_V5 | T14,T15 |
| invalid | Quantity<1 | 0 | Q\_IV1 | T11 |
| invalid | Quantity >999.99 | 1000 | Q\_IV2 | T12 |

Table 2 is the test cases for item ID and quantity by using Equivalence classes and boundary value analysis. **Note that we use Robust Boundary Value Analysis method.**

Table 2. Test case 1

|  |  |
| --- | --- |
| **Test case number** | **T1** |
| Tags covered | ID\_V1, Q\_Normal |
| **Input** | |
| Item ID | 0 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | 1 |
| Item total | 10 |
| Cart contents | Item id:0 Quantity 10 |
| Cart total | 10 |

Table 3. Test case 2

|  |  |
| --- | --- |
| **Test case number** | **T2** |
| Tags covered | ID\_V2, Q\_Normal |
| **Input** | |
| Item ID | 1 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | 1.1 |
| Item total | 11 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10 |
| Cart total | 21 |

Table 4. Test case 3

|  |  |
| --- | --- |
| **Test case number** | **T3** |
| Tags covered | ID\_V3, Q\_Normal |
| **Input** | |
| Item ID | 99998 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | 200.00 |
| Item total | 2000 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:99998 Quantity 10 |
| Cart total | cart total is invalid |

Table 5. Test case 4

|  |  |
| --- | --- |
| **Test case number** | **T4** |
| Tags covered | ID\_V4, Q\_Normal |
| **Input** | |
| Item ID | 99999 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | 400.00 |
| Item total | 4000.00 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:99999 Quantity 10 |
| Cart total | cart total is invalid |

Table 6. Test case 5

|  |  |
| --- | --- |
| **Test case number** | **T5** |
| Tags covered | ID\_IV1, Q\_Normal |
| **Input** | |
| Item ID | -1 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | Valid |
| Item total | Valid |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id(-1) is valid |
| Cart total | Item id is valid |

Table 7. Test case 6

|  |  |
| --- | --- |
| **Test case number** | **T6** |
| Tags covered | ID\_IV2, Q\_Normal |
| **Input** | |
| Item ID | 10000 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | Valid |
| Item total | Valid |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id(10000) is valid |
| Cart total | Item id is valid |

Table 8. Test case 7

|  |  |
| --- | --- |
| **Test case number** | **T7** |
| Tags covered | ID\_ Normal, Q\_V1 |
| **Input** | |
| Item ID | 100 |
| Quantity | 1 |
| **Expected Results** | |
| Item price | 10 |
| Item total | 10 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id100 Quantity 1 |
| Cart total | 31 |

Table 9. Test case 8

|  |  |
| --- | --- |
| **Test case number** | **T8** |
| Tags covered | ID\_ Normal, Q\_V2 |
| **Input** | |
| Item ID | 100 |
| Quantity | 2 |
| **Expected Results** | |
| Item price | 20 |
| Item total | 20 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id100 Quantity 2 |
| Cart total | 41 |

Table 10. Test case 9

|  |  |
| --- | --- |
| **Test case number** | **T9** |
| Tags covered | ID\_ Normal, Q\_V3 |
| **Input** | |
| Item ID | 100 |
| Quantity | 98 |
| **Expected Results** | |
| Item price | 10 |
| Item total | 980 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id100 Quantity 98 |
| Cart total | cart total is invalid(>999.99) |

Table 11. Test case 10

|  |  |
| --- | --- |
| **Test case number** | **T10** |
| Tags covered | ID\_ Normal, Q\_V4 |
| **Input** | |
| Item ID | 100 |
| Quantity | 99 |
| **Expected Results** | |
| Item price | 10 |
| Item total | 990 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:100 Quantity 99 |
| Cart total | cart total is invalid(>999.99) |

Table 12. Test case 11

|  |  |
| --- | --- |
| **Test case number** | **T11** |
| Tags covered | ID\_ Normal, Q\_IV1 |
| **Input** | |
| Item ID | 100 |
| Quantity | 0 |
| **Expected Results** | |
| Item price | 10 |
| Item total | 0 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10 |
| Cart total | 21 |

Table 13. Test case 12

|  |  |
| --- | --- |
| **Test case number** | **T12** |
| Tags covered | ID\_ Normal, Q\_IV2 |
| **Input** | |
| Item ID | 100 |
| Quantity | 1000 |
| **Expected Results** | |
| Item price | 10 |
| Item total | 10000 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:100 Quantity is invalid |
| Cart total | Quantity is invalid |

Table 14. Test case 13

|  |  |
| --- | --- |
| **Test case number** | **T13** |
| Tags covered | ID\_ Normal, Q\_ Normal |
| **Input** | |
| Item ID | 100 |
| Quantity | 10 |
| **Expected Results** | |
| Item price | 10 |
| Item total | 100 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:100 Quantity 10 |
| Cart total | 121 |

Table 15. Test case 14

|  |  |
| --- | --- |
| **Test case number** | **T14** |
| Tags covered | ID\_V5, Q\_V5 |
| **Input** | |
| Item ID | 5000 |
| Quantity | 40 |
| **Expected Results** | |
| Item price | 21.9745 |
| Item total | 878.98 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:5000 Quantity 40 |
| Cart total | 999.98 |

Table 16. Test case 15

|  |  |
| --- | --- |
| **Test case number** | **T15** |
| Tags covered | ID\_V6, Q\_V5 |
| **Input** | |
| Item ID | 5001 |
| Quantity | 50 |
| **Expected Results** | |
| Item price | 21.9475 |
| Item total | 878.99 |
| Cart contents | Item id:0 Quantity 10  Item id:1 Quantity 10  Item id:5001 Quantity 40 |
| Cart total | 999.99 |

# Testing additional condition or variable

Table 17 is the condition table for item Item Price

Table 17. Condition table for Item price

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Class type** | **Class** | **Boundary** | **Tag** | **TC** |
| **Item Price** | valid | [1-400] | 10 | P\_Normal | T7 |
| valid | [1-400] | 1 | P\_V1 | T1 |
| valid | [1-400] | 1.1 | P\_V2 | T2 |
| valid | [1-400]] | 399 | P\_V3 | T3 |
| valid | [1-400] | 400 | P\_V4 | T4 |
| invalid | Price <1 | 0 | P\_IV1 | T5 |
| invalid | Price>400 | 401 | P\_IV2 | T6 |

Table 18. Test case 1

|  |  |
| --- | --- |
| **Test case number** | **T1** |
| Tags covered | P\_V1 |
| **Input** | |
| Item Price | 1 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | 2 |

Table 19. Test case 2

|  |  |
| --- | --- |
| **Test case number** | **T2** |
| Tags covered | P\_V2 |
| **Input** | |
| Item Price | 1.1 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | 2.2 |

Table 20. Test case 3

|  |  |
| --- | --- |
| **Test case number** | **T3** |
| Tags covered | P\_V3 |
| **Input** | |
| Item Price | 399 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | 798 |

Table 21. Test case 4

|  |  |
| --- | --- |
| **Test case number** | **T4** |
| Tags covered | P\_V4 |
| **Input** | |
| Item Price | 400 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | 400 |

Table 22. Test case 5

|  |  |
| --- | --- |
| **Test case number** | **T5** |
| Tags covered | P\_V5 |
| **Input** | |
| Item Price | 0 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | Price is invalid |

Table 23. Test case 6

|  |  |
| --- | --- |
| **Test case number** | **T6** |
| Tags covered | P\_V6 |
| **Input** | |
| Item Price | 401 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | Price is invalid |

Table 24. Test case 7

|  |  |
| --- | --- |
| **Test case number** | **T7** |
| Tags covered | P\_Normal |
| **Input** | |
| Item Price | 10 |
| Quantity | 2 |
| **Expected Results** | |
| Item Total | 20 |

# 2. Task 2

# 2.1 Test case design

Table 25 is the condition table for length 、Input Type and Repeat.

Table 25. Condition table for length 、Input Type and Repeat

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Class type** | **Class** | **Boundary** | **Tag** | **TC** |
| **length** | valid | 81 | 81 | L\_Normal | T5-T10 |
| invalid | Length=0 | 0 | L\_IV1 | T1 |
| invalid | Length=1 | 1 | L\_IV2 | T2 |
| invalid | Length<81 | 80 | L\_IV3 | T3 |
| invalid | Length>81 | 82 | L\_IV4 | T4 |
| **Type** | valid | number | 1..9 | T\_Normal | T10 |
| invalid | letter | a..z  A..Z | T\_IV1 | T5 |
| invalid | Special char | ,.;’:!@  ‘ ’“... | T\_IV2 | T6 |
| **Repeat** | valid | norepeat | 1348… | R\_Normal | T10 |
| invalid | rowrepeat | 1134… | R\_IV1 | T7 |
| invalid | colrepeat | 1234..  1452.. | R\_IV2 | T8 |
| invalid | gridrepeat | 123..  456..  189.. | R\_IV3 | T9 |

Table 26 is the test cases for length 、Input Type and Repeat by using Equivalence classes and boundary value analysis. **Note that we use Robust Boundary Value Analysis method.**

Table 26. Test case 1

|  |  |
| --- | --- |
| **Test case number** | **T1** |
| Tags covered | L\_IV1 |
| **Input** | |
| Input numbers | None |
| **Expected Results** | |
| Return result | **Input is None** |

Table 27. Test case 2

|  |  |
| --- | --- |
| **Test case number** | **T2** |
| Tags covered | L\_IV2 |
| **Input** | |
| Input numbers | 1 |
| **Expected Results** | |
| Return result | The string variable isn’t 81 characters long |

Table 28. Test case 3

|  |  |
| --- | --- |
| **Test case number** | **T3** |
| Tags covered | L\_IV3 |
| **Input** | |
| Input numbers | 41736982563215894795872431682543716979158643234691  275828964357157329168416487529(contents 80 numbers) |
| **Expected Results** | |
| Return result | The string variable isn’t 81 characters long |

Table 29. Test case 4

|  |  |
| --- | --- |
| **Test case number** | **T4** |
| Tags covered | L\_IV4 |
| **Input** | |
| Input numbers | 41736982563215894795872431682543716979158643234691  27582896435715732916841648752935(contents 82 numbers) |
| **Expected Results** | |
| Return result | The string variable isn’t 81 characters long |

Table 30. Test case 5

Program returns integer code based on the validity of the provided Sudoku solution:

Return 1 = solution contains characters which are not a number.

|  |  |
| --- | --- |
| **Test case number** | **T5** |
| Tags covered | L\_Normal、T\_IV1 |
| **Input** | |
| Input numbers | 417yhgkn5632158gsg9587243168ijdfh16979158643234691  2758289643571573291684164sjxb9(contents letter) |
| **Expected Results** | |
| Return result | 1 |

Table 31. Test case 6

Program returns integer code based on the validity of the provided Sudoku solution:

Return 1 = solution contains characters which are not a number.

|  |  |
| --- | --- |
| **Test case number** | **T6** |
| Tags covered | L\_Normal、T\_IV2 |
| **Input** | |
| Input numbers | @！￥%%……%……\*@#￥￥%……\*（）《》？：：“”：？？|？+——-=-=|………  (contents Special characters) |
| **Expected Results** | |
| Return result | 1 |

Table 32. Test case 7

Program returns integer code based on the validity of the provided Sudoku solution:

R3: A digit can appear only once in the rows of the global grid.

Return -3 = solution is violating R3

|  |  |
| --- | --- |
| **Test case number** | **T7** |
| Tags covered | L\_Normal、R\_IV1 |
| **Input** | |
| Input numbers | 1342...  1678...  (The number repeats on the same line) |
| **Expected Results** | |
| Return result | **-3** |

Table 33 Test case 8

Program returns integer code based on the validity of the provided Sudoku solution:

R4: A digit can appear only once in the columns of the global grid.

Return -4 = solution is violating R4

|  |  |
| --- | --- |
| **Test case number** | **T8** |
| Tags covered | L\_Normal、R\_IV2 |
| **Input** | |
| Input numbers | 123..  189..  (The number repeats on the same column) |
| **Expected Results** | |
| Return result | **-4** |

Table 34 Test case 9

Program returns integer code based on the validity of the provided Sudoku solution:

R2: All digits appear only once in a sub-grid, i.e. they cannot repeat

Return -2 = solution is violating R2

|  |  |
| --- | --- |
| **Test case number** | **T9** |
| Tags covered | L\_Normal、R\_IV3 |
| **Input** | |
| Input numbers | 123..  456..  189..  (The number repeats on the same grid) |
| **Expected Results** | |
| Return result | **-2** |

Table 35 Test case 10

|  |  |
| --- | --- |
| **Test case number** | **T910** |
| Tags covered | L\_Normal、T\_Normal、R\_Normal |
| **Input** | |
| Input numbers | 417369825632158947958724316825437169791586432346912758289643571573291684164875293 |
| **Expected Results** | |
| Return result | **0** |

# 2.2 Test codes

|  |
| --- |
| **public** **class** SudokuVerifier {  //sudoku checker modified from the original code by Ibrahim Ali  //https://codereview.stackexchange.com/a/180840    **public** **static** **final** **int** ***LENGTH*** = 81;  **public** **static** **final** **int** ***ROW\_LENGTH*** = 9;  **public** **int** verify(String candidateSolution) {  **int**[][] sudoku = **new** **int**[***ROW\_LENGTH***][***ROW\_LENGTH***];  **char**[] arrayChars;  **int** sudokuRowSize = 0;  **int** sudokuColumnSize = 0;    **if** (!lengthChecker(candidateSolution)) {  **return** -1;  } **else** {  arrayChars = candidateSolution.toCharArray();  **for** (**int** loop = 0; loop < arrayChars.length; loop++) {  **try** {  **int** number = Integer.*parseInt*("" + arrayChars[loop]);  **if** (number > 0 && number <= 9) {  **if** (loop > 0 && loop % 9 == 0) {  sudokuColumnSize = 0;  sudokuRowSize++;  }  sudoku[sudokuRowSize][sudokuColumnSize] = number;  sudokuColumnSize++;  } **else** {  **return** -1;  }  } **catch** (NumberFormatException ne) {  ne.printStackTrace();  **return** 1;  }  }  **return** sudokuRuleChecker(sudoku);  }  }  //checks whether if any of the rules are broken (columns, rows, sub-grids).  **private** **int** sudokuRuleChecker(**int**[][] sudoku) {  **int** result = 0;  **for** (**int** i = 0; i < ***ROW\_LENGTH***; i += 3) {  **for** (**int** j = 0; j < ***ROW\_LENGTH***; j += 3) {  **if** (duplicateChecker(gridToArrayConverter(i, j, sudoku))) {  **return** -2;  }  }  }  **for** (**int** loop = 0; loop < ***ROW\_LENGTH***; loop++) {  **if** (duplicateChecker(sudoku[loop])) {  **return** -3;  }  }  **for** (**int** loop = 0; loop < ***ROW\_LENGTH***; loop++) {  **int**[] arrayToCheck = **new** **int**[9];  **for** (**int** loop2 = 0; loop2 < ***ROW\_LENGTH***; loop2++) {  arrayToCheck[loop2] = sudoku[loop2][loop];  }  **if** (duplicateChecker(arrayToCheck)) {  **return** -4;  }  }  **return** result;  }    //checks whether a single dimensional array has any duplicates  **public** **boolean** duplicateChecker(**int**[] arrayToCheck) {  **boolean** result = **false**;  **for** (**int** loop = 0; loop < ***ROW\_LENGTH***; loop++) {  **int** numberToCheck = arrayToCheck[loop];  **for** (**int** innerLoop = 0; innerLoop < ***ROW\_LENGTH***; innerLoop++) {  **if** (innerLoop != loop && numberToCheck == arrayToCheck[innerLoop]) {  result = **true**;  **break**;  }  }  }  **return** result;  }    //Converts two dimensional array to a single dimensional array  **public** **int**[] gridToArrayConverter(**int** i, **int** j, **int**[][] array) {  **int**[] result = **new** **int**[***ROW\_LENGTH***];  **int** resultIndex = 0;  **for** (**int** k = i; k < (i + 3); k++) {  **for** (**int** l = j; l < (j + 3); l++) {  result[resultIndex] = array[k][l];  resultIndex++;  }  }  **return** result;  }  // checks whether String is LENGTH characters long  **public** **boolean** lengthChecker(String stringToCheck) {  **boolean** result = **false**;  **if** (stringToCheck.length() == ***LENGTH***) {  result = **true**;  }  **return** result;  }  /\* noted by phf  public static void main(String[] args) {  SudokuVerifier sudokuObject = new SudokuVerifier();  int result = sudokuObject  .verify("417369825632158947958724316825437169791586432346912758289643571573291684164875293");  System.out.println("Result: " + result);  }  \*/  } |