Number Range Summarizer

Lightweight Technical Documentation

January 29, 2021

A. Troskie
troskie.a@gmail.com

Title: Number Range Summarizer
Creation date: January 29, 2021

File history:

Date	Author	Description
January 29, 2021	Anton Troskie	Document creation

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Program Operation

Implement code which has the ability to produce a comma delimited list of numbers, grouping numbers into a range when appear to be sequential. An example is depicted in Code Snippet 1.

Further requirements are:

1. Program must be written entirely in Java; minimum version 8.
2. Solution must be uploaded to Github.
3. Solution must implement the provided interface.
4. Solution must have valid unit tests.

```
Please enter a comma delimited integer array:
1,3,6,7,8,12,13,14,15,21,22,23,24,31

Input: 1, 3, 6, 7, 8, 12, 13, 14, 15, 21, 22, 23, 24, 31
Output: 1, 3, 6-8, 12-15, 21-24, 31

Process finished with exit code 0
```

Code Snippet 1: An example of usage.

1.2 Assumptions and Requirements

We assume that the input data may contain a various degrees of error in terms of illegal characters, incorrect delimiter usage and/or spacing, as well as the repetition of identical integers.

These issues should be addressed accordingly, and the program should be able to:

1. Detect the usage of illegal characters.
2. Automatically correct the format of user input in the case of:
   - Multiple instances of the same integer.
   - Scattered spacing between integer and delimiter pairs.
   - Accidental placement of multiple delimiters in succession.

2 Implementation

2.1 Parsing User Input

To successfully parse the user input, all degrees of error should be taken into account. The function `collect` as seen in Code Snippet 2 takes a string as an argument and will return a populated collection of integers depending on the content of the input string.

Additionally, the `collect` function makes use `formatInput` and `parseInput` to operate correctly.

```java
public Collection<Integer> collect(String input) {
        input = formatInput(input);
        if (input != null) {
                return parseInput(input);
        } else {
                return null;
        }
}
```

Code Snippet 2: *collect* function.

The `formatInput` function as shown in Code Snippet 3 is responsible for correctly formatting user input in the case of misaligned spacing, multiple consecutive delimiters, and the presence of multiple identical integers. In the event that illegal characters are present, an empty string will be returned. Table 2 provides an overview of the `formatInput` function.

```java
private String formatInput(String input) {
        input = input.replaceAll("\\s+", "");
        while (input.contains(",,")) {
                input = input.replaceFirst(",,", ",");
        }
        for (int i = 0; i < input.length(); i++) {
                char a = input.charAt(i);
                if (!(((a >= '0') && (a <= '9')) || (a == ',') || (a == '
                 ↪  '))) {
                        return "";
                } else if (i == 0 && a == ',') {
                        input = input.substring(1);
                        i--;
                } else if (i == input.length() - 1 && a == ',') {
                        input = input.substring(0, input.length());
                }
        }
        return input;
}
```

Code Snippet 3: *formatInput* function.

Table 2: Overview of *formatInput* function.

Line #	**Description**
2	Remove all spaces contained within the input string.
3-5	Remove all unnecessary/accidental delimiters that appear in succession.
8-9	Check to see if the input string contains any illegal characters.
10-12	If the first character of the input string contains a delimiter, remove it.
13-15	If the last character of the input string contains a delimiter, remove it.

The `parseInput` function as shown in Code Snippet 4 is responsible for parsing the return value of `formatInput`, as well as removing any integers that have identical counterparts. This function will return an ordered collection of integers. Table 4 provides an overview of the `parseInput` function.

```
 1  private List<Integer> parseInput(String input) {
 2          int placeHolder = 0;
 3          List<Integer> tempList = new ArrayList<>();
 4          for (int i = 0; i < input.length(); i++) {
 5                  if (input.charAt(i) == ',') {
 6                          Integer tempValue =
                            ↪  parseInt(input.substring(placeHolder, i));
 7                          tempList.add(tempValue);
 8                          placeHolder = i + 1;
 9                  } else if (i == input.length() - 1) {
10                          tempList.add(parseInt(input.substring(placeHolder,
                            ↪  i + 1)));
11                  }
12          }
13          List<Integer> result = new ArrayList<>();
14          for (Integer i : tempList) {
15                  if (!result.contains(i)) result.add(i);
16          }
17          Collections.sort(result);
18          return result;
19  }
```

Code Snippet 4: *parseInput* function.

Table 4: Overview of *parseInput* function.

Line #	**Description**
2-12	Convert input string to a collection of integers.
13-16	Remove all duplicates.
17	Sort the collection of integers in ascending order.

2.2 Creating Output

To create the an output string from the return value of `parseInput`, the `summarizeCollection` function as seen in Code Snippet 5 is used. An overview of this function is given in Table 6.

```java
 1 public String summarizeCollection(Collection<Integer> input) {
 2         StringBuilder result = new StringBuilder();
 3         List<Integer> tempList = new ArrayList<>();
 4         int[] tempArray = new int[input.size()];
 5         boolean sequentialVal = false;
 6         int tempCount = 0;
 7         for (Integer i : input) {
 8                 tempArray[tempCount] = i;
 9                 tempCount++;
10         }
11         for (int i = 0; i < tempArray.length; i++) {
12                 if (i == tempArray.length - 1) {
13                         result.append(tempArray[i]);
14                         break;
15                 } else if (tempArray[i + 1] - tempArray[i] == 1) {
16                         if (!sequentialVal) {
17                                 result.append(tempArray[i]).append("-");
18                         }
19                         sequentialVal = true;
20                 } else {
21                         result.append(tempArray[i]).append(", ");
22                         sequentialVal = false;
23                 }
24         }
25         return result.toString();
26 }
```

Code Snippet 5: *summarizeCollection* function.

Table 6: Overview of *summarizeCollection* function.

Line #	**Description**
7-10	Convert collection of integers to primitive integer array.
11-24	Format output string.

3 Evaluation

3.1 Error Detection and Format Auto-correction

To see if the program correctly handles faulty user input, both `testHandleCharacters` as well as `testAutoFormatInput` tests are performed. These tests are respectively shown in Code Snippet 6 and 7, with an overview of each test provided in Table 8 and 10.

```
1  @Test
2  @DisplayName("Test to see if unacceptable characters or wrong delimiters
    ↪ are handled.")
3  void testHandleCharacters() {
4          Summarizer testSummarizer = new Summarizer();
5          Assertions.assertEquals(Collections.EMPTY_LIST,
            ↪ testSummarizer.collect("1;2;3;4;5;6;7;8"));
6          Assertions.assertEquals(Collections.EMPTY_LIST,
            ↪ testSummarizer.collect("1,f,3,-4,5,6.7,7,8"));
7  }
```

Code Snippet 6: *testHandleCharacters* test.

```
1  @Test
2  @DisplayName("Test to see if misaligned spacing, multiple consecutive
    ↪ delimiters, and multiple instances of the same value are handled.")
3  void testAutoFormatInput() {
4          Summarizer testSummarizer = new Summarizer();
5          List<Integer> testList = Arrays.asList(1,2,3,5,9);
6          Assertions.assertEquals(testList, testSummarizer.collect("1,   2,
            ↪ ,3   ,5,   9"));
7          Assertions.assertEquals(testList, testSummarizer.collect(",1,   2,
            ↪ ,3   ,5,,9,"));
8          Assertions.assertEquals(testList, testSummarizer.collect("1,1,1,
            ↪ 2,2       ,3   ,5,   9"));
9  }
```

Code Snippet 7: *testAutoFormatInput* test.

Table 8: Overview of *testHandleCharacters* test.

Line #	**Description**
5	Test to see if incorrect delimiters are detected.
6	Test to see if illegal characters are detected.

Table 10: Overview of *testAutoFormatInput* test.

Line #	Description
6	Test to see if misaligned spacing is appropriately handled.
7	Test to see if accidental duplicate delimiters are appropriately handled.
8	Test to see if identical integer values that occur more than once are appropriately handled.

3.2 Output Inspection

To determine if the program generally functions as expected, the `testCorrectOutput` test as shown in Code Snippet 8 is performed. This test will evaluate the output of the program with respect to all the possible cases of user input. An overview of this test is shown in Table 12.

```java
@Test
@DisplayName("Test to see if the correct output is obtained.")
void testCorrectOutput() {
        Summarizer testSummarizer = new Summarizer();
        String testOutput = "1, 3, 6-8, 12-15, 21-24, 31";
        Assertions.assertEquals(testOutput,
         ↪  testSummarizer.summarizeCollection(testSummarizer.collect
         ↪  ("1,3,6,7,8,12,13,14,15,21,22,23,24,31")));
        Assertions.assertEquals(testOutput,
         ↪  testSummarizer.summarizeCollection(testSummarizer.collect
         ↪  ("1 ,  3,  6,  7, 8,  12,  13,  14  ,15, 21,  22,23,24,31")));
        Assertions.assertEquals(testOutput,
         ↪  testSummarizer.summarizeCollection(testSummarizer.collect
         ↪  (",1,, 3,6   ,,7,8,,  12,13,  ,,14,15,21  ,,22,23, 24,31,
         ↪  ")));
        Assertions.assertEquals(testOutput,
         ↪  testSummarizer.summarizeCollection(testSummarizer.collect
         ↪  (",1,1,1,, 3,6   ,,7,7,7,7,8,,  12,13,  ,,14,15,15,15,21
         ↪  ,,22,23, 24,31, ")));
        Assertions.assertEquals("",
         ↪  testSummarizer.summarizeCollection(testSummarizer.collect
         ↪  ("1;3;6;7;8;12;13;14;15;21;22;23;24;31")));
        Assertions.assertEquals("",
         ↪  testSummarizer.summarizeCollection(testSummarizer.collect
         ↪  ("1,f,6,7.2,8,12,13,14,-15,21,22,23,24,31")));
}
```

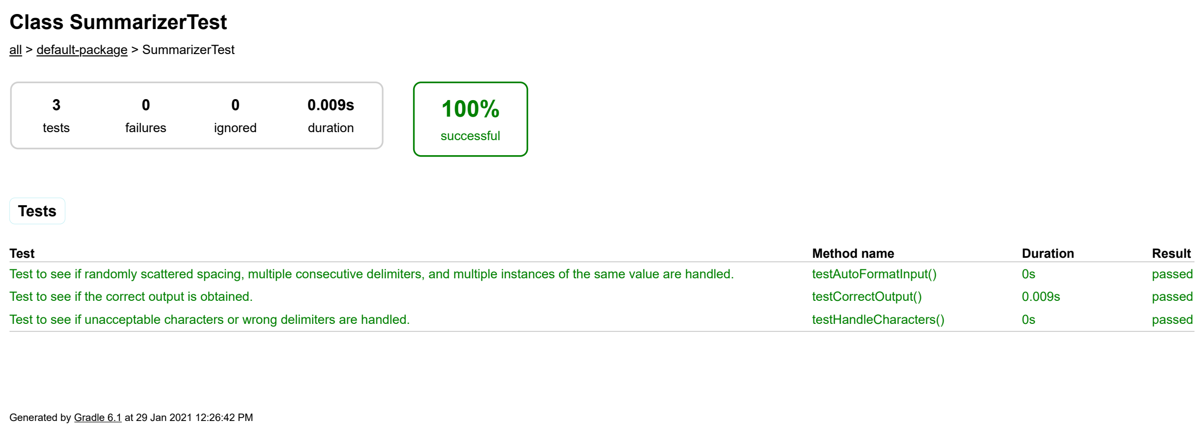
Code Snippet 8: *testCorrectOutput* test.

Table 12: Overview of *testCorrectOutput* test.

Line #	**Description**
6	Test to see if the correct output is obtained from appropriate user input.
7	Test to see if the correct output is obtained from user input with misaligned spacing.
8	Test to see if the correct output is obtained from user input with misaligned spacing as well as the presence of multiple delimiters in succession.
9	Test to see if the correct output is obtained from user input with misaligned spacing, the presence of multiple delimiters in succession, as well as the presence of multiple instances of the same value.
10	Test to see if the correct output is obtained from user input with incorrect delimiter.
11	Test to see if the correct output is obtained from user input with illegal characters.

3.3 Test Results

The results obtained from the unit tests are displayed in Figure 1. It is clear from these results that the program performs as required.



Class SummarizerTest

all > default-package > SummarizerTest

3	0	0	0.009s	100%
tests	failures	ignored	duration	successful

Tests

Test	Method name	Duration	Result
Test to see if randomly scattered spacing, multiple consecutive delimiters, and multiple instances of the same value are handled.	testAutoFormatInput()	0s	passed
Test to see if the correct output is obtained.	testCorrectOutput()	0.009s	passed
Test to see if unacceptable characters or wrong delimiters are handled.	testHandleCharacters()	0s	passed

Generated by Gradle 6.1 at 29 Jan 2021 12:26:42 PM

Figure 1: Unit test results.