Test Driven Development Walkthrough

R. M. J. Chathuranga Madawala
17209393

String Calculator Requirements

- The method can take 0, 1 or 2 numbers separated by a comma(,); for example “” or “1” or “1,2”
- For an empty string, it will return 0
- Method will return their sum
- Allow the add method to handle an unknown amount of numbers
- Allow the add method to handle new lines between numbers (instead of commas). The following input is ok: “1\n2,3” (will equal 6)

String Calculator Requirements

- Support different delimiters
  - To change a delimiter, the beginning of the string will contain a separate line that looks like this: “//[delimiter]\n[numbers…]” for example “//;\n1;2” should return three where the default delimiter is ‘;’ .
  - The first line is optional. All existing scenarios should still be supported.
- Calling add with a negative number will throw an exception “negatives not allowed” – and the negative that was passed.
- Numbers bigger than 1000 should be ignored, so adding 2 + 1001 = 2

Requirement 4

Allow the add method to handle an unknown amount of numbers

Step 1: Update the Test Cases

- Dropped Test Cases
  - More than 2 numbers: “1,2,3” (throws RuntimeException)
- New Test Cases:
  - 3 numbers: “1,2,3” (returns 6)

Step 2: Update Unit Tests

```
Run Test | Debug Test
@Test
Run Test | Debug Test
public final void unkAmountOfNumbers(){
    assertEquals(1+2+3, StringCalculator.add("1,2,3"));
}
```

Step 4: Run the Updated Tests

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - FAILS
  - Negative Tests:
    - `nonNumberUsed()` - PASSES

Step 3a: Update the Implementation

```java
public class StringCalculator {
    public static final int add(String numbers){

        if(numbers.isEmpty()) return 0;

        int runningTotal = 0;
        String[] numbersArray = numbers.split(",");

        for(String number : numbersArray){

            runningTotal += Integer.parseInt(number);
        }

        return runningTotal;
```

```
if(numbersArray.length > 2){
throw new RuntimeException("Up to 2 numbers seperated by a comma");
}
```
REMOVED

Step 4b: Run the tests again

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
  - Negative Tests:
    - `nonNumberUsed()` - PASSES

Requirement 5

Allow the add method to handle new lines between numbers (instead of commas). The following input is ok: “1\n2,3” (will equal 6)

Step 1: Update the Test Cases

- All Existing Test Cases are still necessary
- New Test Cases:
  - 3 numbers inc. new delimiter: “1\n2,3” (returns 6)

Step 2: Update Unit Tests

```
Run Test | Debug Test
@Test
Run Test | Debug Test
public final void newLineDelimiterAnd3Numbers( ){
    assertEquals(1+2+3, StringCalculator.add("1\n2,3"));
}
```

Step 4: Run the Updated Tests

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) - FAILS
  - Negative Tests:
    - `nonNumberUsed()` - PASSES

Step 3a: Update the Implementation

```
public class StringCalculator {
    public static final int add(String numbers){

        if(numbers.isEmpty()) return 0;

        int runningTotal = 0;
        String[] numbersArray = numbers.split(",|\n");

        for(String number : numbersArray){

            runningTotal += Integer.parseInt(number);
        }

        return runningTotal;
```

String[] numbersArray = numbers.split(",|\n");

Step 4b: Run the tests again

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) - PASSES
  - Negative Tests:
    - `nonNumberUsed()` - PASSES

Requirement 6

Support different delimiters

- To change a delimiter, the beginning of the string will contain a separate line that looks like this: “//[delimiter]\n[numbers…]” for example “//;\n1;2” should return three where the default delimiter is ‘;’ .
- The first line is optional. All existing scenarios should still be supported.

Step 1: Update the Test Cases

- All Existing Test Cases are still necessary
- New Test Cases:
  - “//;\n1;2” two numbers including a change in delimiter (returns 3)

Step 2: Update Unit Tests

```
Run Test | Debug Test
@Test
Run Test | Debug Test
public final void changeInDelimiters(){
    assertEquals(1+2, StringCalculator.add("//;\n1;2"));
}
```

Step 4: Run the Updated Tests

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) – PASSES
    - changeInDelimiters() - FAILS
  - Negative Tests:
    - `nonNumberUsed()` - PASSES

Step 3a: Update the Implementation

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class StringCalculator {
    public static final int add(String numbers){

        if(numbers.isEmpty()) return 0;

        Pattern pattern = Pattern.compile("(/{2}.\\n)");
        Matcher matcher = pattern.matcher(numbers);

        String delimiter = ",|\n";
        if(matcher.find()){
            String delimiterString = matcher.group(1);
            delimiter = delimiterString.substring(2,3);
            numbers = numbers.substring(4);
        }

        int runningTotal = 0;
        String[] numbersArray = numbers.split(delimiter);

        for(String number : numbersArray){

            runningTotal += Integer.parseInt(number);
        }

        return runningTotal;
    }
}
```

Step 4b: Run the tests again

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) – PASSES
    - changeInDelimiters() - PASSES
  - Negative Tests:
    - `nonNumberUsed()` - PASSES

Requirement 7

Calling add with a negative number will throw an exception “negatives not allowed” – and the negative that was passed.

Step 1: Update the Test Cases

- All Existing Test Cases are still necessary
- New Test Cases:
  - Calling with a negative number (throw an exception “negatives not allowed” – and the negative that was passed.)

Step 2: Update Unit Tests

```
@Test(expected=RuntimeException.class)
Run Test | Debug Test
public final void NegativeNumberUsed(){
    StringCalculator.add("3,-2");
}
```

Step 4: Run the Updated Tests

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) – PASSES
    - changeInDelimiters() - PASSES
  - Negative Tests:
    - `nonNumberUsed()` – PASSES
    - NegativeNumberUsed() - FAILS

Step 3a: Update the Implementation

```
        int runningTotal = 0;
        String[] numbersArray = numbers.split(delimiter);

        for(String number : numbersArray){

            int nextNum = Integer.parseInt(number);
            if(nextNum <0)
            throw new RuntimeException("Negatives not allowed" + nextNum);
            runningTotal += nextNum;

        }

        return runningTotal;
    }
}
```

Step 4b: Run the tests again

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) – PASSES
    - changeInDelimiters() - PASSES
  - Negative Tests:
    - `nonNumberUsed()` – PASSES
    - NegativeNumberUsed() - PASSES

Requirement 8

Numbers bigger than 1000 should be ignored, so adding 2 + 1001 = 2

Step 1: Update the Test Cases

- All Existing Test Cases are still necessary
- New Test Cases:
  - One numbers > 1000 (return 0)
  - Many numbers >1000 each - eg: 1001,1030, 2008(return 0)
  - Mixed sequence - 1, 2, 1001, 3, 2001 (return 6)

Step 2: Update Unit Tests

```
Run Test | Debug Test
@Test
Run Test | Debug Test
public final void oneNumberGreaterThan1000(){
    assertEquals(0, StringCalculator.add("1001"));
}

Run Test | Debug Test
@Test
Run Test | Debug Test
public final void manyNumbersGreaterThan1000(){
    assertEquals(0, StringCalculator.add("1001,1030,2008"));
}

Run Test | Debug Test
@Test
Run Test | Debug Test
public final void mixedWithGreaterThan1000(){
    assertEquals(1+2+3, StringCalculator.add("1,2,1001,3,2001"));
}
```

Step 4: Run the Updated Tests

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) – PASSES
    - changeInDelimiters() – PASSES
    - oneNumberGreaterThan1000() - FAILS
    - manyNumbersGreaterThan1000() - FAILS
    - mixedWithGreaterThan1000() - FAILS
  - Negative Tests:
    - `nonNumberUsed()` – PASSES
    - NegativeNumberUsed() - PASSES

Step 3a: Update the Implementation

```
    int runningTotal = 0;
    String[] numbersArray = numbers.split(delimiter);

    for(String number : numbersArray){

        int nextNum = Integer.parseInt(number);
        if(nextNum <0)
        throw new RuntimeException("Negatives not allowed" + nextNum);

        if(nextNum <= 1000)
        runningTotal += nextNum;

    }

    return runningTotal;
}
```

Step 4b: Run the tests again

- Expected Output:
  - Positive Tests:
    - `twoNumbersUsed()` – PASSES
    - `oneNumberUsed()` – PASSES
    - `noNumberUsed()` – PASSES
    - unkAmountOfNumbers() - PASSES
    - newLineDelimiterAnd3Numbers( ) – PASSES
    - changeInDelimiters() – PASSES
    - oneNumberGreaterThan1000() - PASSES
    - manyNumbersGreaterThan1000() - PASSES
    - mixedWithGreaterThan1000() - PASSES
  - Negative Tests:
    - `nonNumberUsed()` – PASSES
    - NegativeNumberUsed() - PASSES