String Calculator Kata TDD

Hello there and welcome to the Incubyte TDD Kata!
This assessment is the first step in our recruiting process and will be followed by two pair programming/discussion sessions.

What We're Looking For

A Software Craftsperson at Incubyte is a person who has a strong commitment to the craft of software development. Someone who is passionate about software, knows her/his tools well and is able to use them effectively to create carefully crafted software. Ultimately, a person who has a strong sense of what it is they are doing and is self-motivated to learn and grow.

TDD is a core practice for all of us at Incubyte. We strongly believe that well-written software, is a lot more valuable for the business and end users, as compared to software that is hacked together (but works!). Through this assessment, we want to evaluate how readable and testable your code is. We want to see the Software Craftsperson in you.

As software developers, searching the internet is something of a necessity and is vital tool for being effective problem solvers. We encourage you to Google away! You can also visit our inspiration page to find some useful talks and references that will help you sail through this assessment.
With that, let's jump right in! Follow the instructions below, take your time to do it well and send us your kata once you're happy with what you've done.

Things To Keep In Mind

1. Host your solution on a **public** GitHub/GitLab repository
2. Follow best practices for TDD. Watch this video to understand TDD better
3. Commit your changes frequently to show how your code evolves with every step of TDD
4. We encourage you to use the programming language and tools you feel most comfortable with
5. Do not rush, take your time, we want to see your best work!
6. Send us the link to your repo using this Google Form once you're happy with what you've done

Problem Statement

1. In a test-first manner, create a simple class named StringCalculator and a method

   int add(String numbers)

   1. The method can take numbers as a string separated by comma and will return their sum (for an empty string, it will return 0). For example:

      Input: ""
      Output: 0

      Input: "1"
      Output: 1

      Input: "1,2"
      Output: 3

   2. Start with the simplest test case of an empty string and move to one & two numbers.
   3. Remember to solve things as simply as possible so that you force yourself to write tests you did not think about
   4. Remember to refactor after each passing test.

2. Allow the *add* method to handle an unknown amount of numbers
3. Allow alphabets to be included with numbers.

   The numeric value for the alphabet would be equal to its position.

   Such as a = 1, b = 2, c = 3 ... y = 25, z = 26.

   For example:

   Input: "1,2,a,c"
   Output: 7 (1 + 2 + 1 + 3)

   Note: Use lowercase alphabets only.

4. Calling *add* with a negative number will throw an exception “Negatives not allowed” - and the negative that was passed.
5. If there are multiple negatives, show all of them in the exception message
6. Numbers bigger than 1000 should be ignored.

   For example:
   Input: "2,1001"
   Output: 2

(We expect students to solve at least up to this point. Everything from 7 onwards is optional)

7. **(Optional)** Allow the *add* method to handle new lines between numbers (instead of commas).

    1. The following is ok:

Input: "1\n2,3"
Output: 6

    2. The following is INVALID input so do not expect it: “1,\n” (no need to write a test for it)

8. **(Optional)** 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:

Input: "//;\n1;2"
Output: 3

Since the default delimiter is ‘;.’

Note: All existing scenarios and tests should still be supported. Don’t use alphabets and numbers as delimiters.

9. **(Optional)** Support odd/even addition: If the beginning of the string starts with 0// then add numbers at odd indices and if it starts with 1// then add numbers at even indices.

Input: "//[***]\n1***2***3"
Output: 6

Note: Existing test cases should still pass.

ALL THE BEST!!