String Calculator Kata

Roy Osherove (with modifications)

Rules

1. Strictly practice TDD: Red, Green, Refactor
2. Clean Code is required:
   - 2.1. Intention-revealing names
   - 2.2. Verb/verb-phrase method names
   - 2.3. Methods should do one thing and be short, with no side-effects
   - 2.4. Methods should contain only one level of abstraction
   - 2.5. Code should read like a top-down narrative
   - 2.6. No unnecessary code
   - 2.7. DRY
   - 2.8. Unit tests that test pieces of the algorithm, not only acceptance level tests.
3. No use of the debugger is allowed.

The Kata

1. Create a simple String calculator with a method **`int Add(string numbers)`**
   - 1.1. The method can take 0, 1 or 2 numbers, and will return their sum (for an empty string it will return 0) for example **`“” or “1” or “1,2”`**
   - 1.2. Start with the simplest test case of an empty string and move to 1 and two numbers
   - 1.3. Remember to solve things as simply as possible so that you force yourself to write tests you did not think about
   - 1.4. Remember to refactor after each passing test
2. Allow the Add method to handle an unknown amount of numbers
3. Allow the Add method to handle new lines between numbers (instead of commas).
   - 3.1. the following input is ok: `“1\n2,3”` (will equal 6)
   - 3.2. the following input is NOT ok: `“1,\n”` (not need to prove it - just clarifying)
4. **Support different delimiters**
   - 4.1. To change a delimiter, the beginning of the string will contain a separate line specifying the custom delimiter. This input looks like this: “//{delimiter}\n{numbers…}” (Note that the curly braces are representing the sections of the input and are not input formatting).
   - 4.2. For example: “//;\n1;2” should return a result of 3 because the delimiter is now ‘;’.
   - 4.3. The first line is optional (all existing scenarios should still be supported).
   - 4.4. Do not worry about supporting the specification of ‘\n’ as an explicit custom delimiter. New lines should always be supported as delimiters in your number string.

5. Calling Add with a negative number will throw an exception “negatives not allowed” - and the negative that was passed, if there are multiple negatives, show all of them in the exception message

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

7. Delimiters can be of any length with the following format: “//[{delimiter}]\n{numbers…}”

    7.1. For example: “//[***]\n1***2***3” should return 6.

    7.2. Note that the square brackets are required around the multiple character delimiter.

    7.3. A square bracket is not a valid delimiter.

8. Allow multiple delimiters like this: “//[{delim1}][{delim2}]\n{numbers…}”

    8.1. For example “//[*][%]\n1*2%3” should return 6.

    8.2. Note that once again the square brackets are required around each custom delimiter.

9. Make sure you can also handle multiple delimiters with length longer than one char