

Exercise Unit Testing  
The String Calculator Kata

For this exercise, we’ll write a solution for Roy Osherov’s [String Calculator kata](https://osherove.com/tdd-kata-1). A coding kata is a programming exercise, meant to allow developers to practice fundamental agile software-engineering practices, such as refactoring, and Test Driven Development.

For simplicity’s sake, we ignore some of the kata’s requirements. What follows are the requirements we’re going to use:

1. We’re going to create a class called StringCalculator, with a single static method with the signature static int Add(string numbers);
2. If we pass an empty string, the method should return zero.
3. Passing a single number should result in the number itself.
4. The method takes a string representing numbers separated by a comma, and returns their sum. “1,8,3” should result in 12.
5. The method should ignore numbers greater than 1000. So, “1,2,1000” should result in 1003, but “1,2,1001” should result in 3.
6. If we pass negative numbers, the method should throw an ArgumentException, with the message “Negative numbers not allowed:” followed by the negatives that were specified.

Create a new .Net Core Class Library project ‘StringCalculatorKata’ and add a second MSTest Test project for .Net Core ‘StringCalculatorKataTest’ to the solution.

The next thing we have to do is to ensure our test project can see our production project. To solve that, go to the solution explorer, right-click the test project, then click on ‘Project reference’. Do not forget to add the ‘Using’ statement on top of the test class calling the production project library.

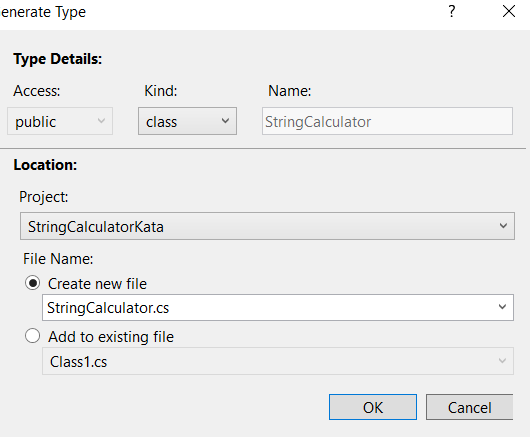
**Starting Our Coding Kata requirement 2**

In our first test case, we test the simplest possible scenario. That is, we call the Add method passing an empty string, which, according to the requirements you saw before, should result in 0.

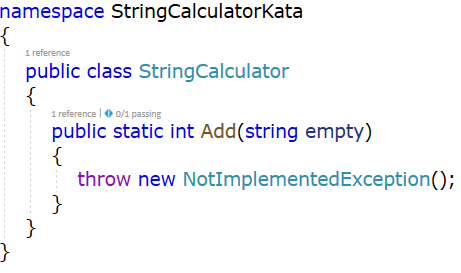
So write this first failing test. Open the StringCalculatorKataTest class and add the test method Add\_EmptyStringAsParam\_ReturnsZero() to it. Organise your code conform the AAA principle.

Of course, neither the Add method nor the StringCalculator class exists, so your code doesn’t even compile.

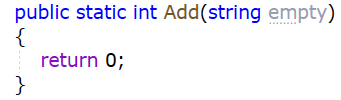
However, you can create the missing StringCalculator class from here. Click on “Show potential fixes” and then on “Generate new type…”. You should then see a window prompting you for the details and location of the new type. Change the “access” to “public” and the location to the production project.



Next error is that the ‘StringCalculator’ does not contain a definition for ‘Add’. Click on Show potential fixes, and then click on “Generate method ‘StringCalculator.Add’”.  
At this point, your complete StringCalculator class should look like this:



Now run the test and off course it will fail and show the ‘NotImplementedException’-message. Now, you are ready to make the test pass. In this case, you can simply make the Add method return zero.



## **Move on to Coding Kata requirement 3**

The requirement says that passing a single number should return the number itself.

First step is coding the test method “Add\_StringContainingSingleNumber\_ReturnsTheNumberItself()”.   
Write your code using the AAA pattern.

Again the test will fail. Rewrite your Add-method in the laziest possible way to make this test and the first one pass.

## **Move on to Coding Kata requirement 4**

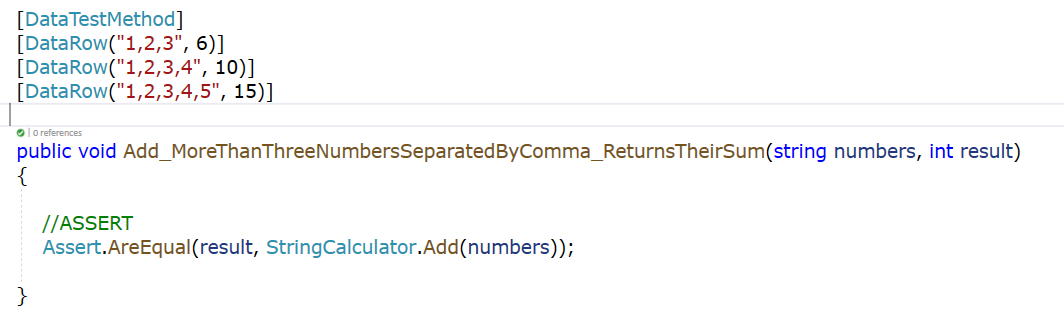
Since we’ve already tested the Add method by passing zero numbers (an empty string) and a single number, it feels like the next natural step to write a test for the scenario of adding two numbers. So code the test Add\_TwoNumbersSeparatedByComma\_ReturnsTheirSum() Run the test and adjust the Add-method. Make sure the first 2 tests don’t fail.

Of course the add method should also be able to sum up a string with 3, 4, 5, … numbers. Write a test method Add\_MoreThanThreeNumbersSeparatedByComma\_ReturnsTheirSum() with a random choice of numbers in the string and adjust the Add method so this test will pass. This time you code the correct implementation of the Add method. Make sure the previous tests don’t fail.

To make sure that the Add method works for a random number of numbers in the input string, it is a good idea to create a parameterized test using the **DataRow attribute.**

The DataRow attribute in MSTest allows setting the values of test parameters. More than one DataRow attribute can be present in the code. All you need to do is replace the TestMethod attribute with the DataTestMethod attribute and pass the test combinations to the test method via DataRows.

In this screenshot, the test will run for each datarow. The first element is the string with the numbers and the second parameter is the expected result. The AAA pattern is now less obvious but still present.



## **Move on to Coding Kata requirement 5**

The requirement of ignoring numbers greater than 1000 can easily be checked with the previous test by just including datarows with numbers bigger than 1000.



The add method needs to be adapted for this extra test to pass.

## **Move on to the final Coding Kata requirement 6**

The requirements says that negative numbers are not allowed.   
For brevity’s sake, just add a single test method with several test cases Add\_StringContainingNegativeNumbers\_ThrowsException().   
For this test, we’re asserting not against a return value. Rather check whether the method under test throws an exception.

Go for the correct Add method implementation right away.

**Need more practice?**

On GitHub (<https://github.com/garora/TDD-Katas>) you find the most famous TDD Katas for practicing your C# skills for Unit testing. You can try the Bowling game Kata or the FizzBuzz kata.

Another catalogue with Katas can be found on  
<https://kata-log.rocks/tdd>