TDD & BDD
Design Through Testing

Exercise 1
String Calculator

Overview

In this exercise we will use the creation of a simple calculator, that takes strings as input representing numbers, and adds them together.

The aim of this exercise is to practice the basic skills of TDD. That is our primary focus, rather than solving the problem - but of course it will be nice to do both.

This is best accomplished by making progress in very small steps - focus on that as you progress through the exercise.

Goals of the Exercise

To practice ***RED, GREEN, REFACTOR!***

To work in small, even tiny, steps.

Exercise

General Advice

Complete each step in order. Don't read ahead beyond the current step that you are working on.

Make progress in very small steps.

Don't rush ahead and solve the problem and forget to test!

Don't add new code, unless you have a failing test that makes you.

Always refactor on a passing test, and run the tests after each, small, change to verify your changes.

Top Tip: Always predict the outcome of your test, even say it out loud, before you run it.

Step 1 - Create a String Calculator

Create a simple String calculator with a method:

> ***int add(string numbers)***

The method can take 0, 1 or 2 numbers, in the form of a string, and will return their sum (for an empty string it will return 0).

For example valid inputs are:

> ***"", "1" or "1,2"***

Advice

Plan your approach, begin with the simplest test case - in this case it's an empty string!

Once that is working, move on to dealing with one number. Once that is working add another test for adding two numbers.

Step 2 - New Delimiters

Now allow the Add method to handle more than two numbers and new lines between numbers (instead of commas) as delimiters.

the following input is ok:

> ***"1\n2,3" (will equal 6)***

Advice

Once again, remember to start with a test. When the problem you are solving offers you clear examples like this one, these examples may be a good place to start.

If your solution can already handle more than 2 numbers, how do you get to a failing test? - Write your "more than 2 numbers" test, then force it to fail, hard code a broken return to see the failure, revert to your original solution and see the test pass.

Step 3 - User Defined Delimiters

Now add a feature to 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" (will equal 3)*** *(the user-defined delimiter is ';')*

The first line is optional. All existing scenarios should still be supported.

Advice

Again, there's an example in the problem description - use that for your test. A word of caution though, always think about whether there are any "gotchas" in the examples presented. (In this case, I don't think there are, but good to be thoughtful about it.)