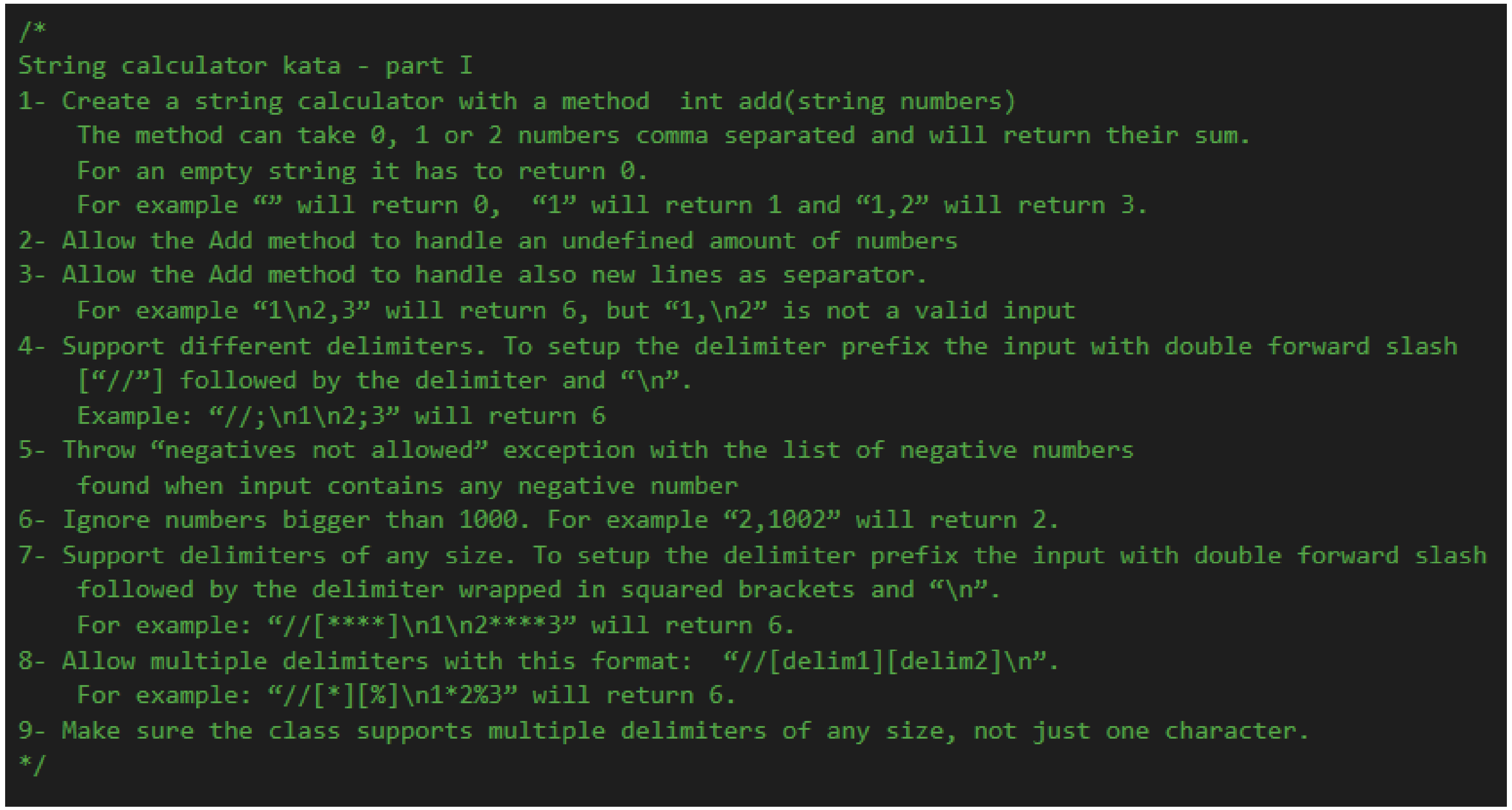
Test-Driven Development

Jérémie Primas
03.03.2022

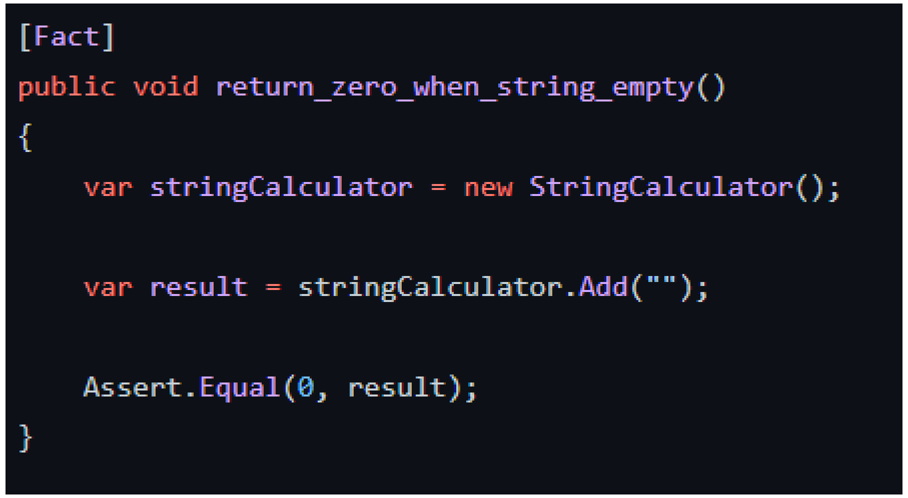
L'exemple du StringCalculator



```
/*
String calculator kata - part I
1- Create a string calculator with a method  int add(string numbers)
    The method can take 0, 1 or 2 numbers comma separated and will return their sum.
    For an empty string it has to return 0.
    For example “” will return 0,  “1” will return 1 and “1,2” will return 3.
2- Allow the Add method to handle an undefined amount of numbers
3- Allow the Add method to handle also new lines as separator.
    For example “1\n2,3” will return 6, but “1,\n2” is not a valid input
4- Support different delimiters. To setup the delimiter prefix the input with double forward slash
    [“//”] followed by the delimiter and “\n”.
    Example: “//;\n1\n2;3” will return 6
5- Throw “negatives not allowed” exception with the list of negative numbers
    found when input contains any negative number
6- Ignore numbers bigger than 1000. For example “2,1002” will return 2.
7- Support delimiters of any size. To setup the delimiter prefix the input with double forward slash
    followed by the delimiter wrapped in squared brackets and “\n”.
    For example: “//[****]\n1\n2****3” will return 6.
8- Allow multiple delimiters with this format:  “//[delim1][delim2]\n”.
    For example: “//[*][%]\n1*2%3” will return 6.
9- Make sure the class supports multiple delimiters of any size, not just one character.
*/
```

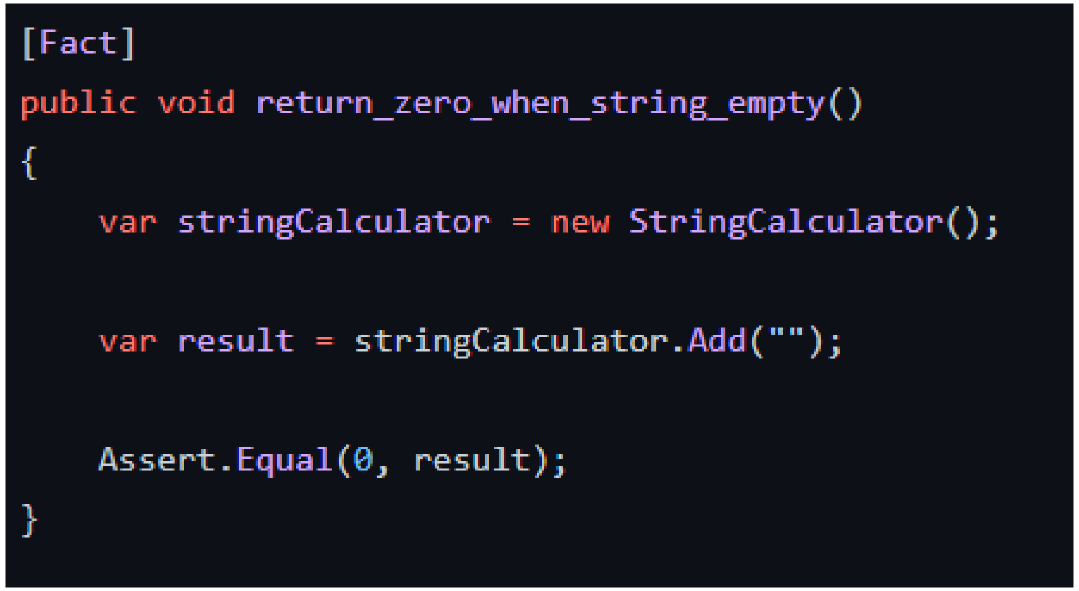
Create a string calculator with a method int add(string numbers)
  The method can take 0, 1 or 2 numbers comma separated and will return their sum.
  For an empty string it has to return 0.
  For example “” will return 0, “1” will return 1 and “1,2” will return 3.

- Baby-step : On teste le cas le plus simple



```
[Fact]
public void return_zero_when_string_empty()
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add("");

    Assert.Equal(0, result);
}
```

Test en rouge

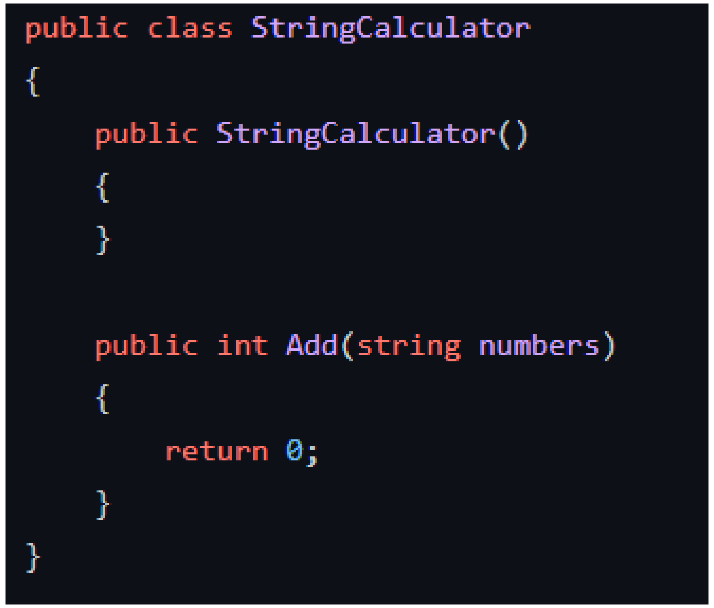


```
[Fact]
public void return_zero_when_string_empty()
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add("");

    Assert.Equal(0, result);
}
```

- Nom explicite
- Séparé en 3 parties : Arrange, Act, Assert
- Ne run pas

"StringCalculator" n'existe pas. Le test ne passe pas.

…qui devient vert

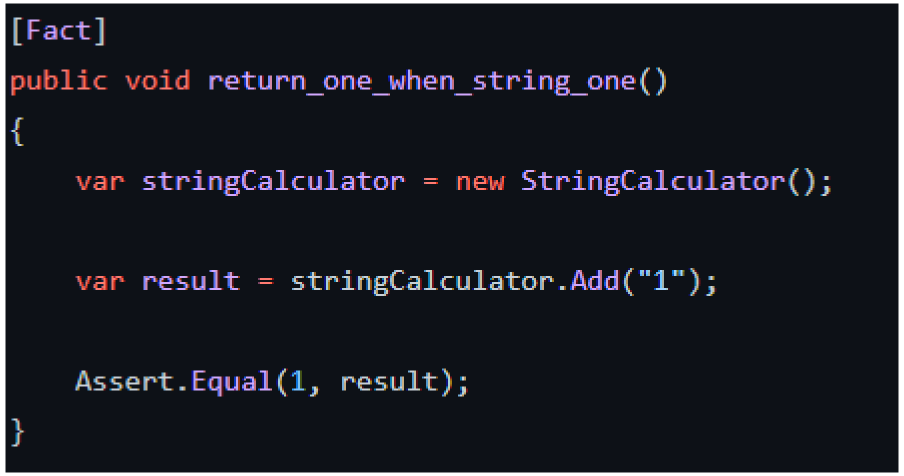


```
public class StringCalculator
{
    public StringCalculator()
    {
    }

    public int Add(string numbers)
    {
        return 0;
    }
}
```

- Fausse implémentation
- Premier exemple de l'implémentation

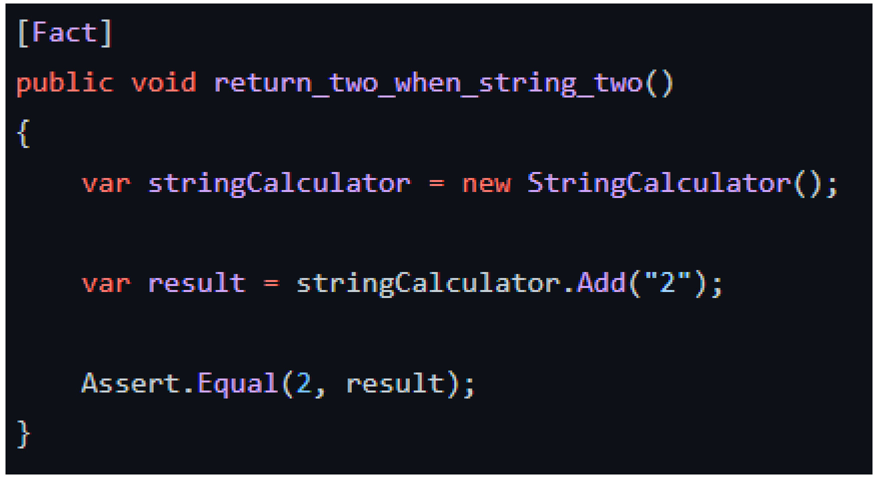
Le test passe au vert.
On oublie pas de commit le changement

Avant de refactorer le code, on ajoute d'autres tests spécifiques triviaux



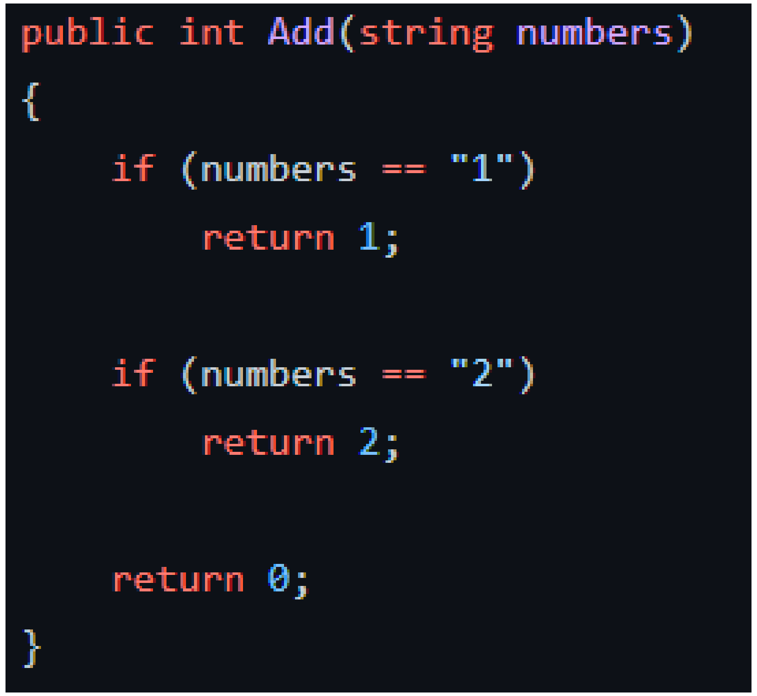
```
[Fact]
public void return_one_when_string_one()
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add("1");

    Assert.Equal(1, result);
}
```

Puis



```
[Fact]
public void return_two_when_string_two()
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add("2");

    Assert.Equal(2, result);
}
```

On run les tests : Rouge
Objectif : Vert

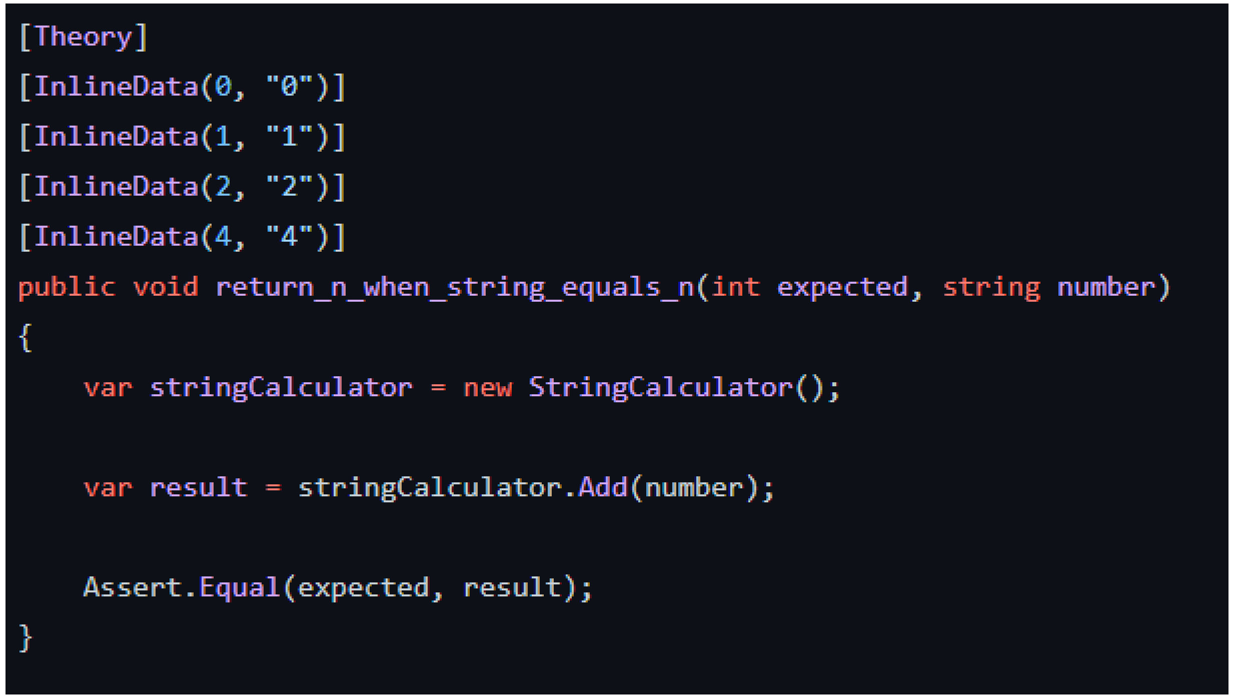


```
public int Add(string numbers)
{
    if (numbers == "1")
        return 1;

    if (numbers == "2")
        return 2;

    return 0;
}
```

- Fausse implémentation

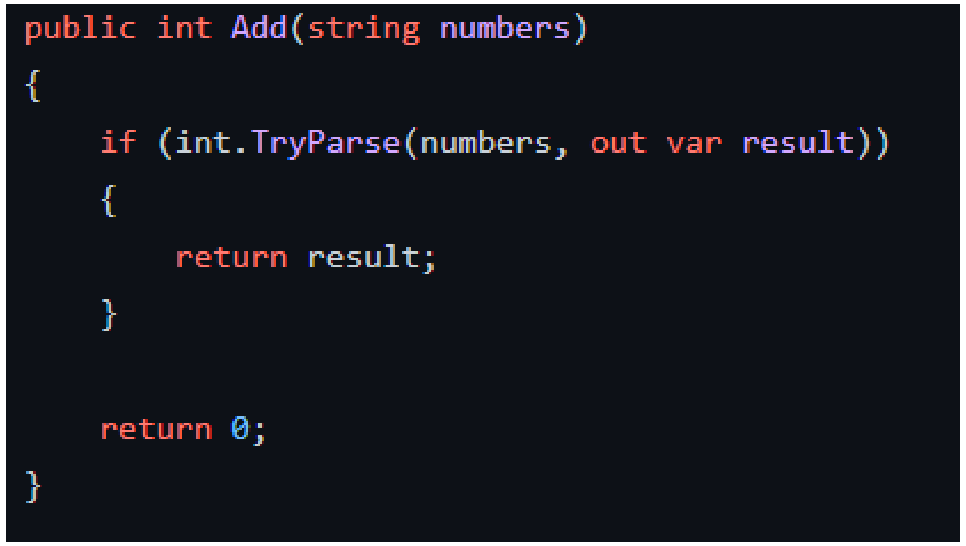
Le test passe à nouveau au vert.
On commit

Simplification de l'écriture du test



```csharp
[Theory]
[InlineData(0, "0")]
[InlineData(1, "1")]
[InlineData(2, "2")]
[InlineData(4, "4")]
public void return_n_when_string_equals_n(int expected, string number)
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add(number);

    Assert.Equal(expected, result);
}
```

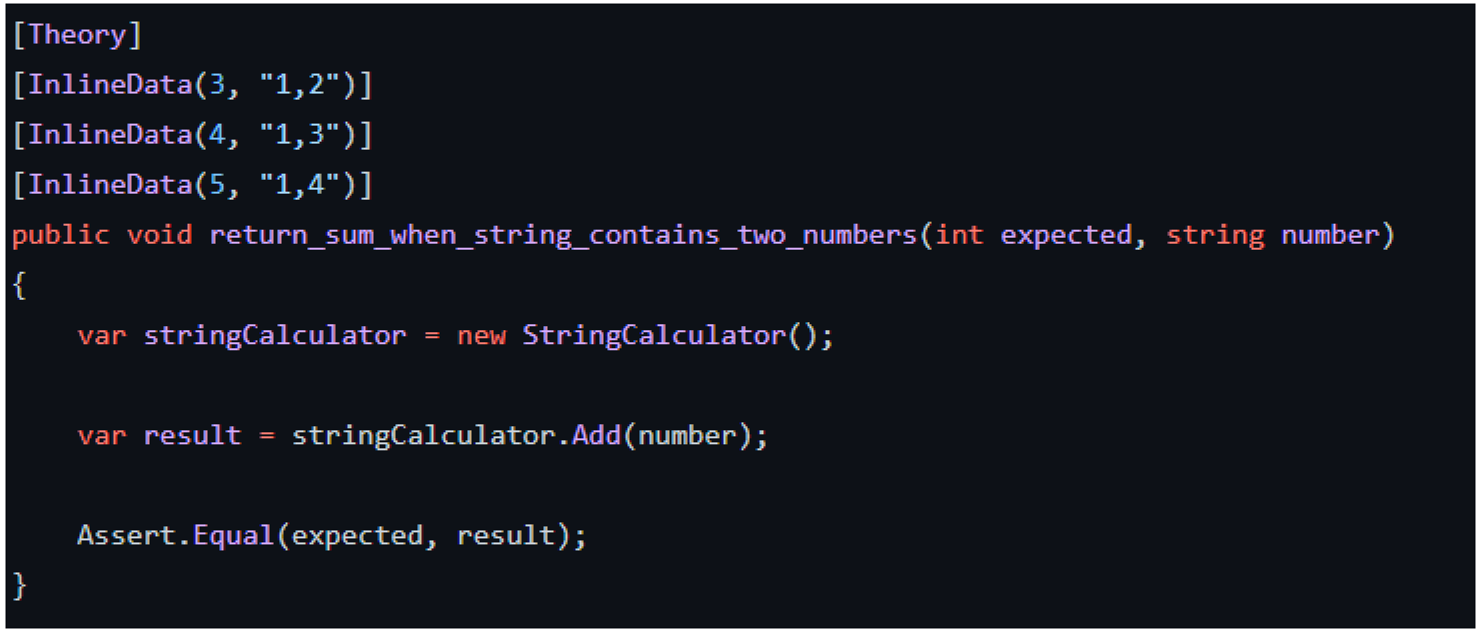
Refacto du code



```csharp
public int Add(string numbers)
{
    if (int.TryParse(numbers, out var result))
    {
        return result;
    }

    return 0;
}
```

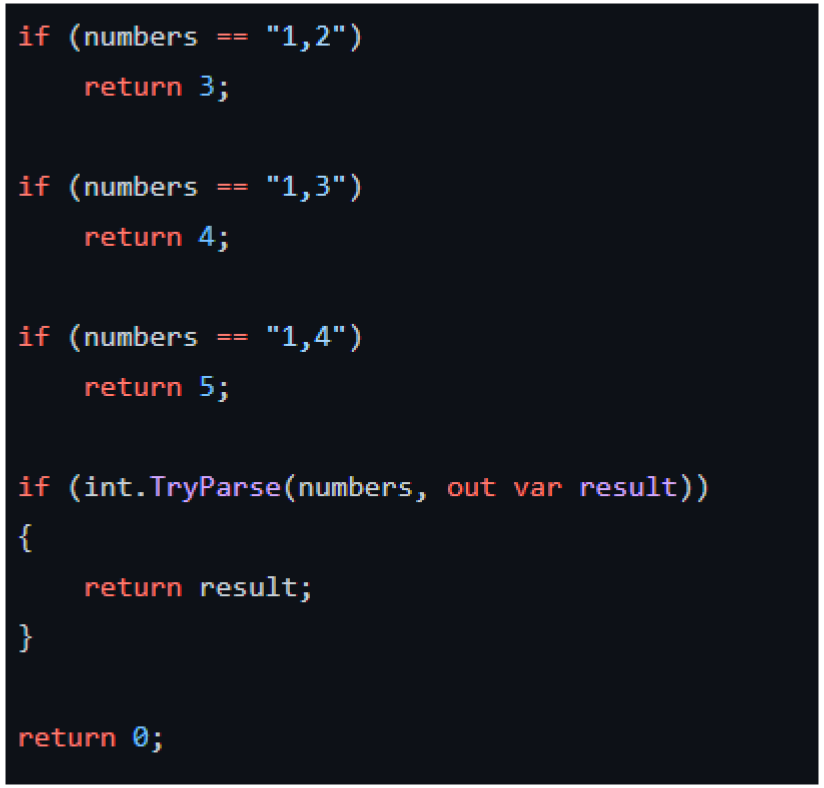
On vérifie si le test tourne encore

Autres cas de test : plusieurs chiffres



```
[Theory]
[InlineData(3, "1,2")]
[InlineData(4, "1,3")]
[InlineData(5, "1,4")]
public void return_sum_when_string_contains_two_numbers(int expected, string number)
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add(number);

    Assert.Equal(expected, result);
}
```

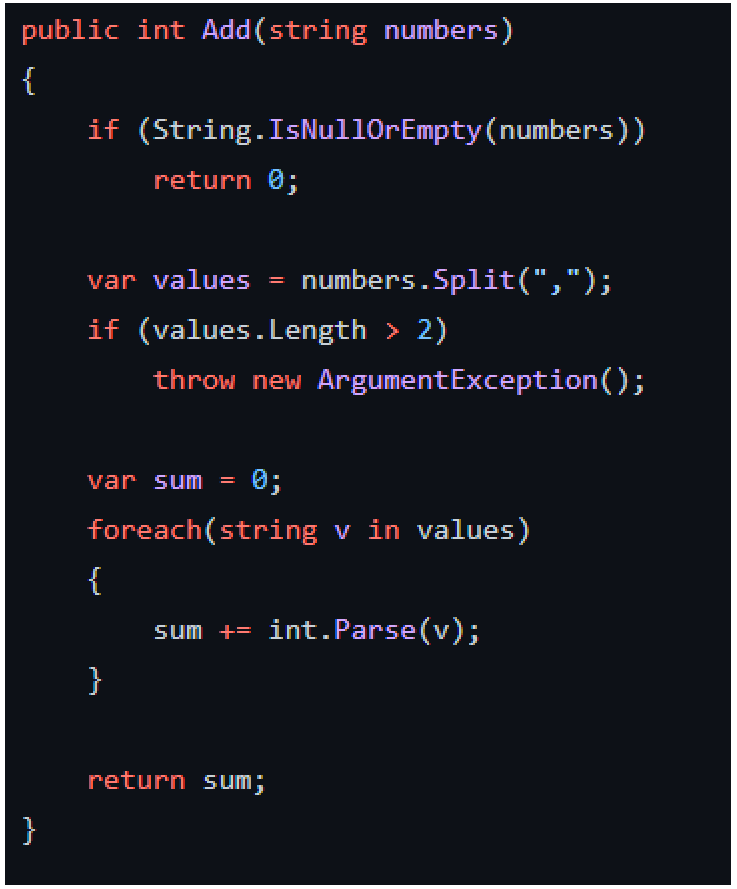
Ces tests passent en rouge



```
if (numbers == "1,2")
    return 3;

if (numbers == "1,3")
    return 4;

if (numbers == "1,4")
    return 5;

if (int.TryParse(numbers, out var result))
{
    return result;
}

return 0;
```

On les fait passer au vert
On commit

On refacto

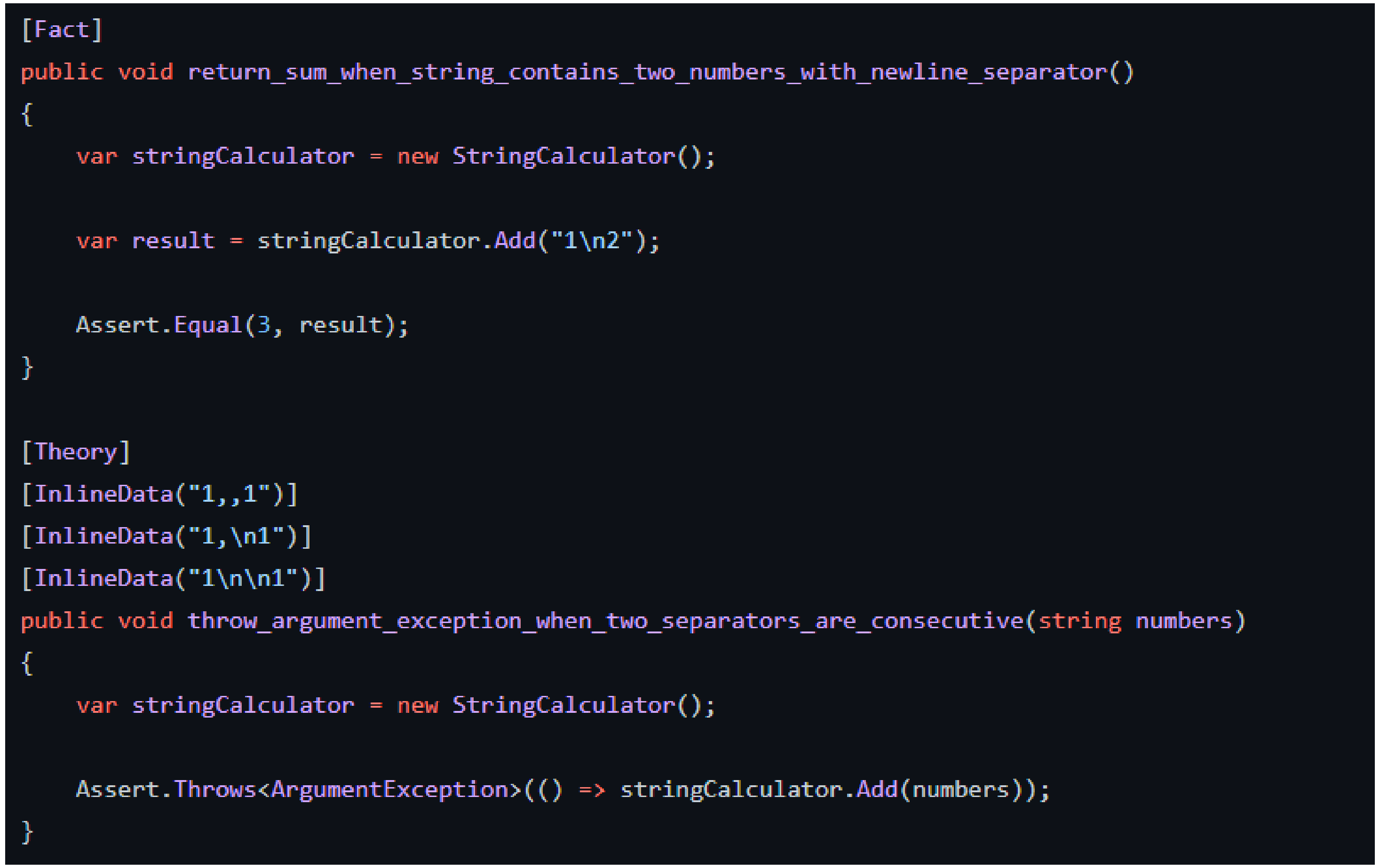


```
public int Add(string numbers)
{
    if (String.IsNullOrEmpty(numbers))
        return 0;

    var values = numbers.Split(",");
    if (values.Length > 2)
        throw new ArgumentException();

    var sum = 0;
    foreach(string v in values)
    {
        sum += int.Parse(v);
    }

    return sum;
}
```

On couvre de plus en plus de cas

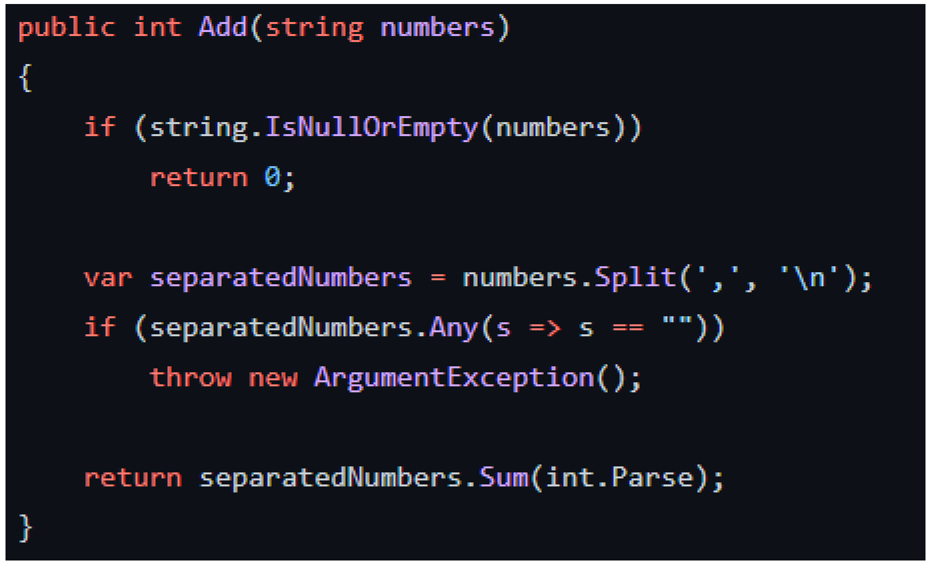
Allow the Add method to handle also new lines as separator.
For example “1\n2,3” will return 6, but “1,\n2” is not a valid input

On continue d'avancer dans la spécification



```
[Fact]
public void return_sum_when_string_contains_two_numbers_with_newline_separator()
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add("1\n2");

    Assert.Equal(3, result);
}

[Theory]
[InlineData("1,,1")]
[InlineData("1,\n1")]
[InlineData("1\n\n1")]
public void throw_argument_exception_when_two_separators_are_consecutive(string numbers)
{
    var stringCalculator = new StringCalculator();

    Assert.Throws<ArgumentException>(() => stringCalculator.Add(numbers));
}
```

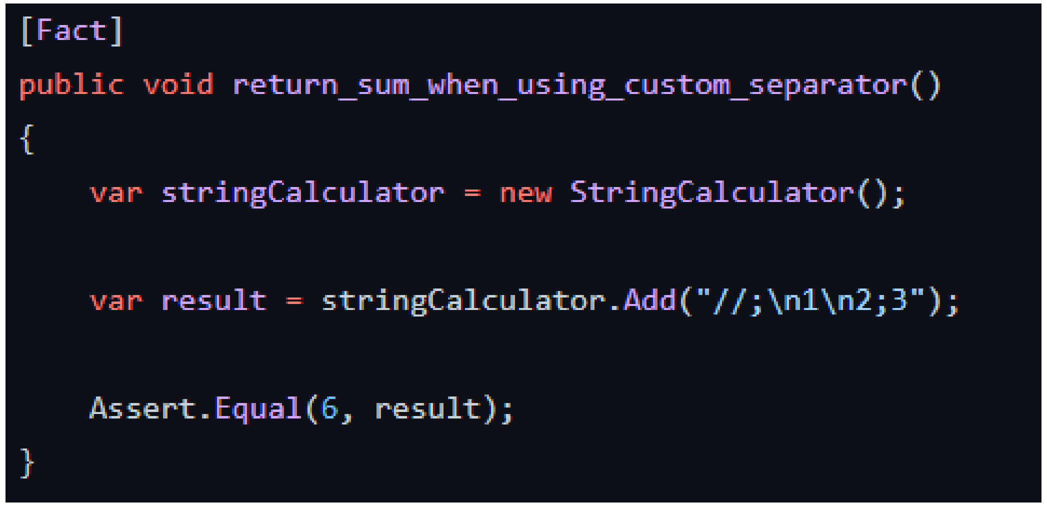
On fait passer les tests au vert



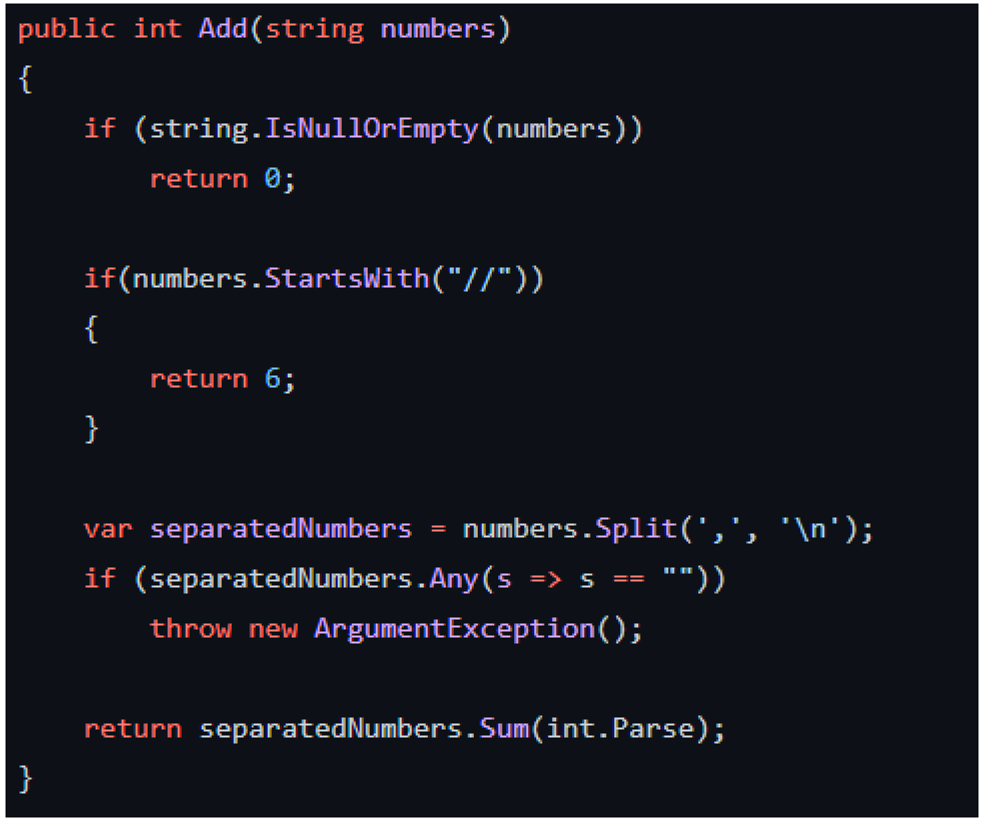
```
public int Add(string numbers)
{
    if (string.IsNullOrEmpty(numbers))
        return 0;

    var separatedNumbers = numbers.Split(',', '\n');
    if (separatedNumbers.Any(s => s == ""))
        throw new ArgumentException();

    return separatedNumbers.Sum(int.Parse);
}
```

Vert + Commit

Support different delimiters. To setup the delimiter prefix the input with double forward slash [“//”] followed by the delimiter and “\n”.
Example: “//;\n1\n2;3” will return 6

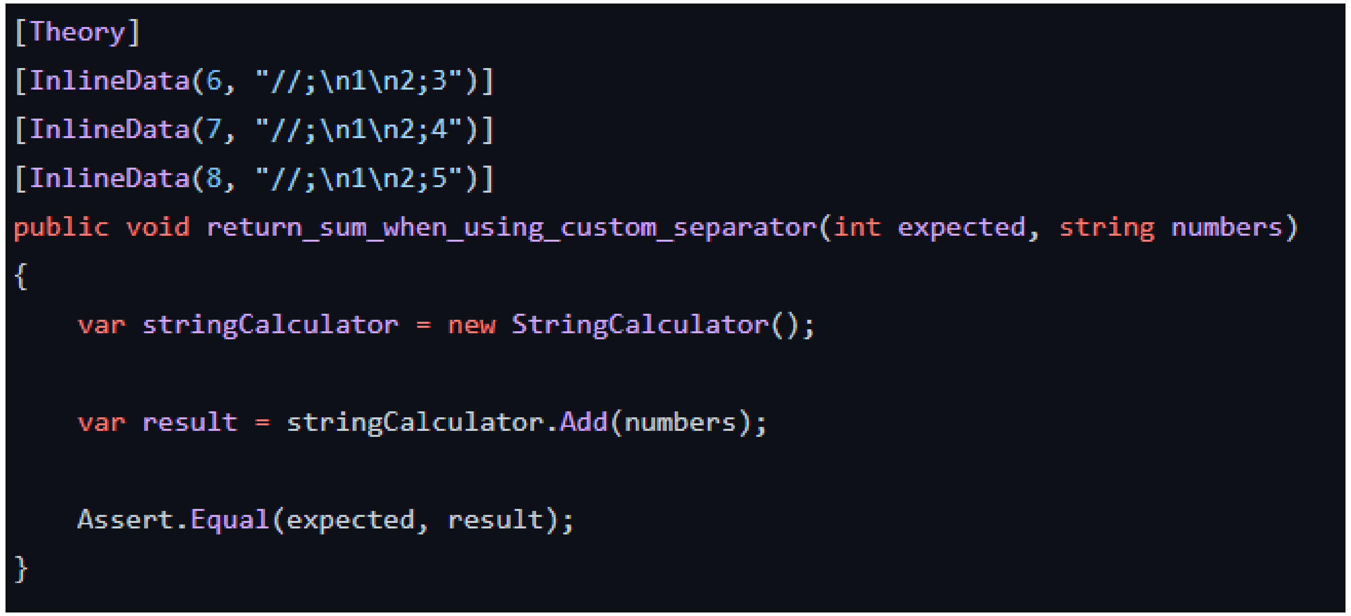


```
[Fact]
public void return_sum_when_using_custom_separator()
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add("//;\n1\n2;3");

    Assert.Equal(6, result);
}
```

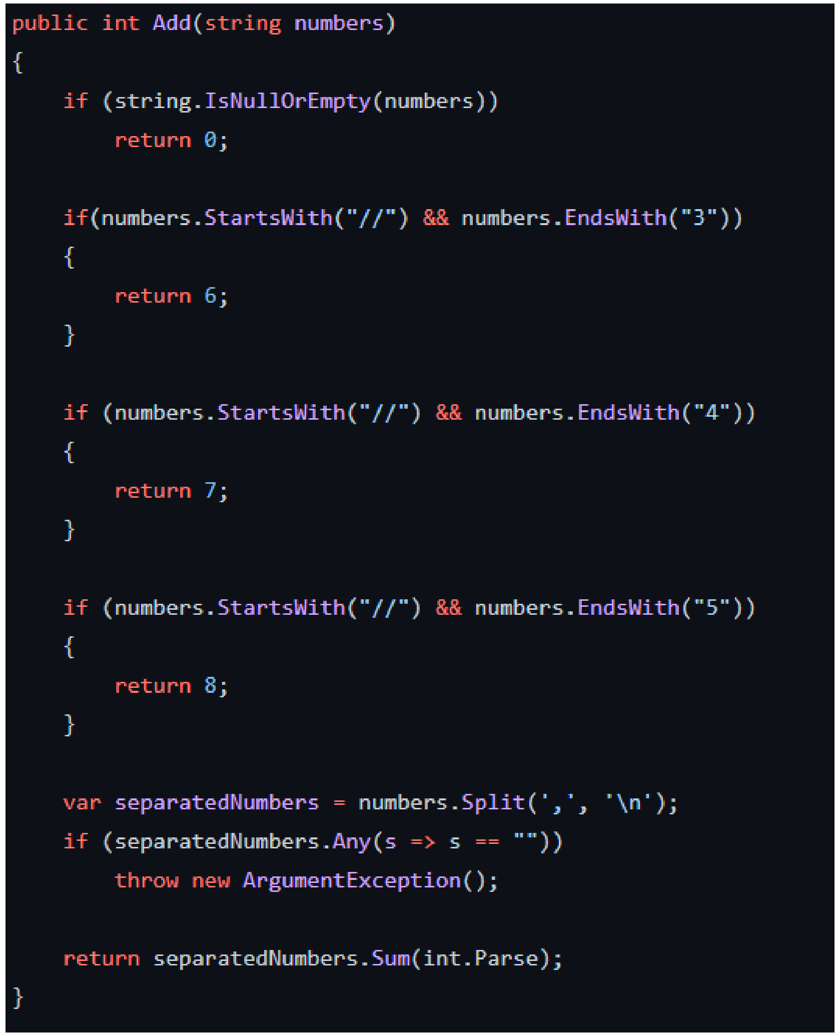


```
public int Add(string numbers)
{
    if (string.IsNullOrEmpty(numbers))
        return 0;

    if(numbers.StartsWith("//"))
    {
        return 6;
    }

    var separatedNumbers = numbers.Split(',', '\n');
    if (separatedNumbers.Any(s => s == ""))
        throw new ArgumentException();

    return separatedNumbers.Sum(int.Parse);
}
```

Support different delimiters. To setup the delimiter prefix the input with double forward slash [“//”] followed by the delimiter and “\n”.
Example: “//;\n1\n2;3” will return 6

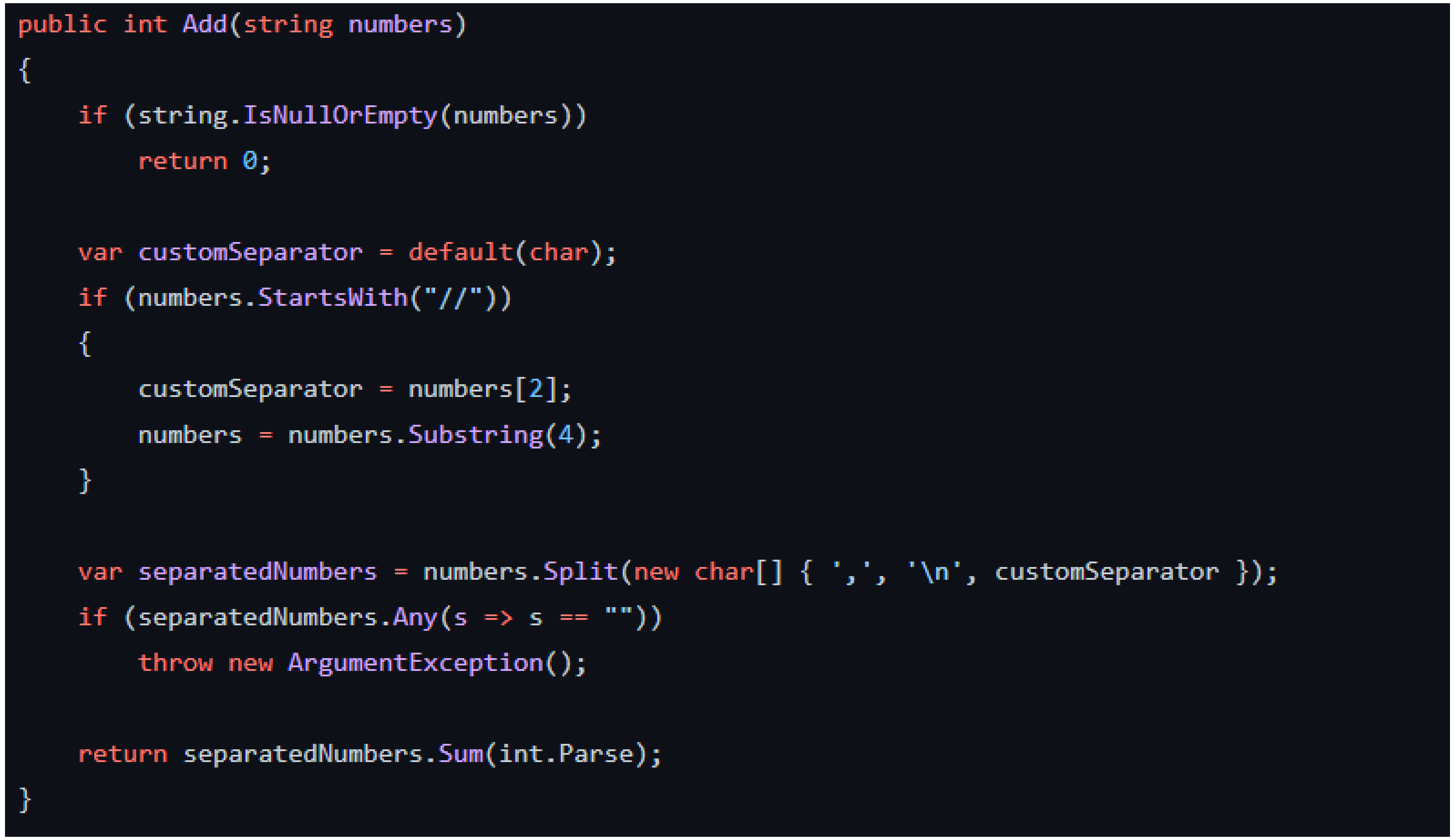


```
[Theory]
[InlineData(6, "//;\n1\n2;3")]
[InlineData(7, "//;\n1\n2;4")]
[InlineData(8, "//;\n1\n2;5")]
public void return_sum_when_using_custom_separator(int expected, string numbers)
{
    var stringCalculator = new StringCalculator();

    var result = stringCalculator.Add(numbers);

    Assert.Equal(expected, result);
}
```



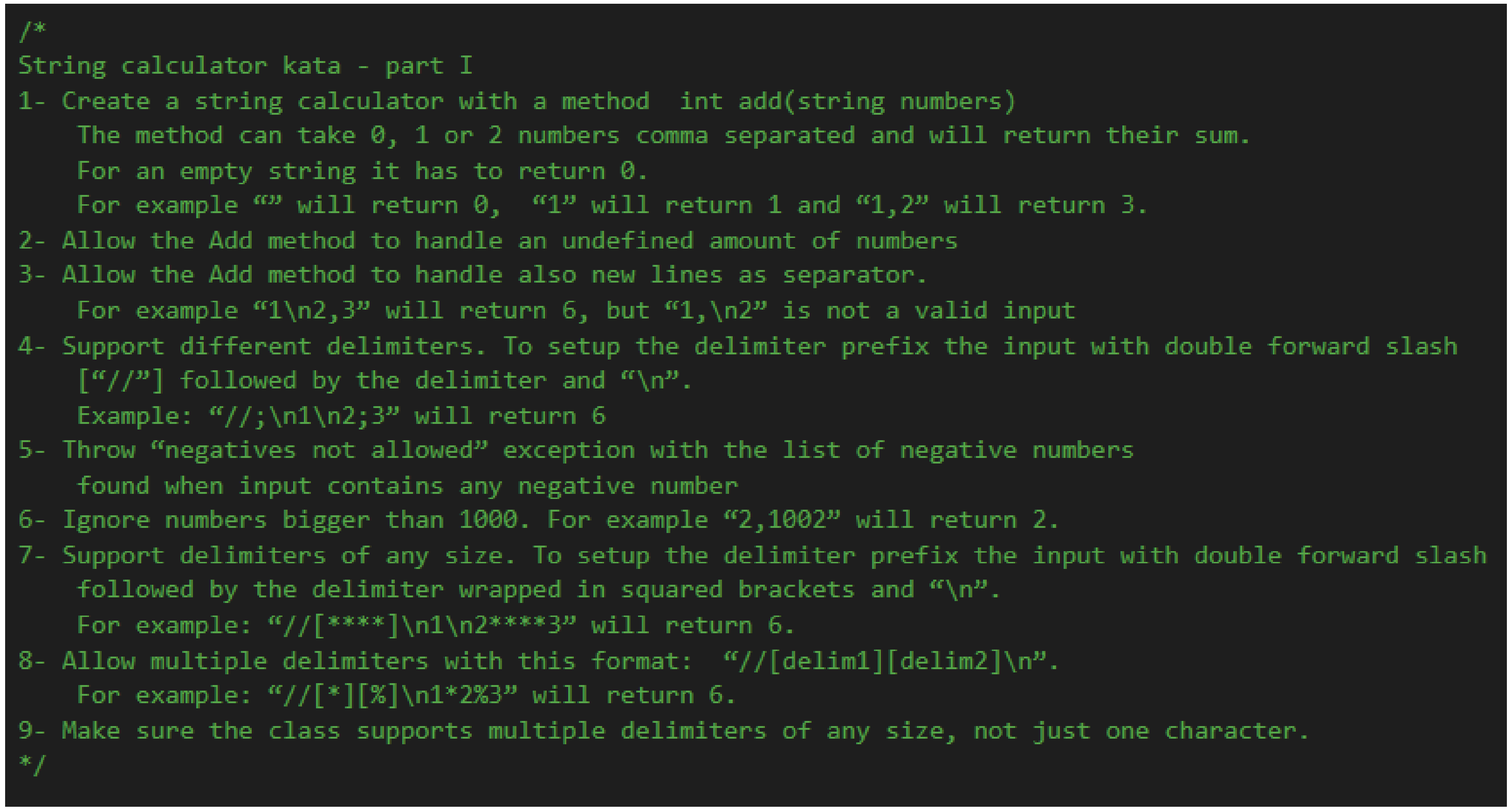
```
public int Add(string numbers)
{
    if (string.IsNullOrEmpty(numbers))
        return 0;

    if(numbers.StartsWith("//") && numbers.EndsWith("3"))
    {
        return 6;
    }

    if (numbers.StartsWith("//") && numbers.EndsWith("4"))
    {
        return 7;
    }

    if (numbers.StartsWith("//") && numbers.EndsWith("5"))
    {
        return 8;
    }

    var separatedNumbers = numbers.Split(',', '\n');
    if (separatedNumbers.Any(s => s == ""))
        throw new ArgumentException();

    return separatedNumbers.Sum(int.Parse);
}
```

Support different delimiters. To setup the delimiter prefix the input with double forward slash [“//”] followed by the delimiter and “\n”.
Example: “//;\n1\n2;3” will return 6



```
public int Add(string numbers)
{
    if (string.IsNullOrEmpty(numbers))
        return 0;

    var customSeparator = default(char);
    if (numbers.StartsWith("//"))
    {
        customSeparator = numbers[2];
        numbers = numbers.Substring(4);
    }

    var separatedNumbers = numbers.Split(new char[] { ',', '\n', customSeparator });
    if (separatedNumbers.Any(s => s == ""))
        throw new ArgumentException();

    return separatedNumbers.Sum(int.Parse);
}
```

L'exemple du StringCalculator



```
/*
String calculator kata - part I
1- Create a string calculator with a method  int add(string numbers)
    The method can take 0, 1 or 2 numbers comma separated and will return their sum.
    For an empty string it has to return 0.
    For example “” will return 0,  “1” will return 1 and “1,2” will return 3.
2- Allow the Add method to handle an undefined amount of numbers
3- Allow the Add method to handle also new lines as separator.
    For example “1\n2,3” will return 6, but “1,\n2” is not a valid input
4- Support different delimiters. To setup the delimiter prefix the input with double forward slash
    [“//”] followed by the delimiter and “\n”.
    Example: “//;\n1\n2;3” will return 6
5- Throw “negatives not allowed” exception with the list of negative numbers
    found when input contains any negative number
6- Ignore numbers bigger than 1000. For example “2,1002” will return 2.
7- Support delimiters of any size. To setup the delimiter prefix the input with double forward slash
    followed by the delimiter wrapped in squared brackets and “\n”.
    For example: “//[****]\n1\n2****3” will return 6.
8- Allow multiple delimiters with this format:  “//[delim1][delim2]\n”.
    For example: “//[*][%]\n1*2%3” will return 6.
9- Make sure the class supports multiple delimiters of any size, not just one character.
*/
```

Le travail par paire et en groupe

Chacun est concentré sur la tâche
Prise d'initiative et du relai

Pilote
Ecrit et explique ce qu'il écrit

Co-pilote
Révise le travail du pilote
Détecte les erreurs accidentelles indétectables
lors du travail seul

Variante : Tour de contrôle et co-co-pilote : Mob programming
Lorsque l'on travail à plusieurs

Merci