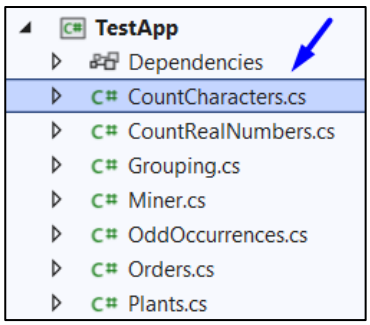
Exercises: Unit Testing Dictionaries

Test your tasks in the Judge system:
https://alpha.judge.softuni.org/contests/dictionaries-lambda-and-linq-unit-testing-exercise/4474

1. Unit Test: Count Characters

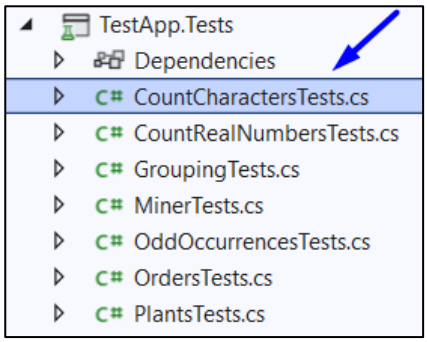
Look at the **provided skeleton** and examine the **CountCharacters.cs** class that you will test:



The method takes in a **list of strings**, and collects the **number of times a character has appeared** and returns a string representing that information:

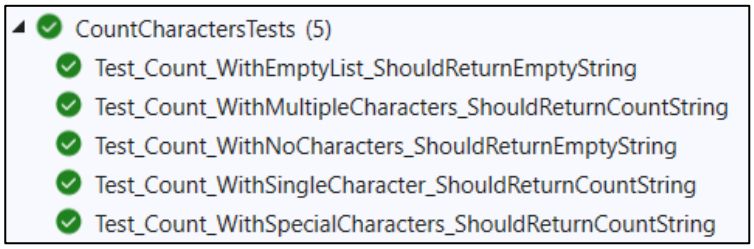
```
public class CountCharacters
{
    5 references | 5/5 passing
    public static string Count(List<string> input)
    {
        Dictionary<char, int> charCount = input.SelectMany(s:string => s)// IEnumerable<char>
            .GroupBy(c:char => c)// IEnumerable<IGrouping<…>>
            .ToDictionary(g:IGrouping<char,char> => g.Key, g:IGrouping<char,char> => g.Count());

        StringBuilder sb = new();
        foreach (KeyValuePair<char, int> pair in charCount)
        {
            sb.AppendLine($"{pair.Key} -> {pair.Value}");
        }

        return sb.ToString().Trim();
    }
}
```

Then, look at the tests inside the **CountCharactersTests.cs** class:



```
public class CountCharactersTests
{
    [Test]
    0 references
    public void Test_Count_WithEmptyList_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_Count_WithNoCharacters_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_Count_WithSingleCharacter_ShouldReturnCountString()...

    [Test]
    0 references
    public void Test_Count_WithMultipleCharacters_ShouldReturnCountString()...

    [Test]
    0 references
    public void Test_Count_WithSpecialCharacters_ShouldReturnCountString()...
}
```

The first test if **finished** so you have a **reference**, **one** is finished **partially**, the rest of the tests are **empty,** and your task is to finish them. The tests should run when you're finished:



2. Unit Test: Count Real Numbers

Test a given method which takes in an **array of integers** and **counts** how many **times** each **number** was seen.

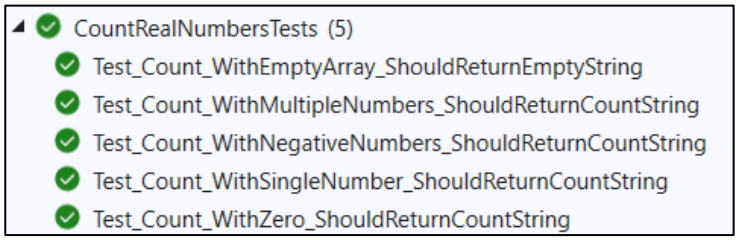
The method is found in the **`CountRealNumbers.cs`** file:

```
public class CountRealNumbers
{
    1 reference
    public static string Count(int[] nums)
    {
        SortedDictionary<int, int> count = new();

        foreach (int num in nums)
        {
            count.TryAdd(num, 0);
            count[num]++;
        }

        StringBuilder sb = new();
        foreach (KeyValuePair<int, int> pair in count)
        {
            sb.AppendLine($"{pair.Key} -> {pair.Value}");
        }

        return sb.ToString().Trim();
    }
}
```

You are given a **test file `CountRealNumbresTests.cs`** which contains **5 tests**. **One** of them has been **finished partially**, and **four** are **empty** for you to finish:

```
public class CountRealNumbersTests
{
    [Test]
    0 references
    public void Test_Count_WithEmptyArray_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_Count_WithSingleNumber_ShouldReturnCountString()...

    [Test]
    0 references
    public void Test_Count_WithMultipleNumbers_ShouldReturnCountString()...

    [Test]
    0 references
    public void Test_Count_WithNegativeNumbers_ShouldReturnCountString()...

    [Test]
    0 references
    public void Test_Count_WithZero_ShouldReturnCountString()...
}
```

When you are ready make sure your **tests run:**



3. Unit Test: Grouping

Test a given method which takes in a **list of integers** and **groups** them by **even** and **odd** numbers.

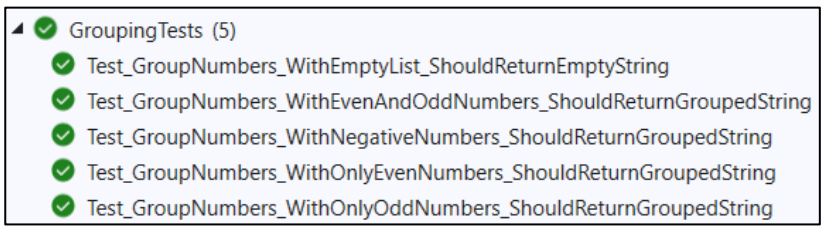
The method is found in the **`Grouping.cs`** file:

```
public class Grouping
{
    5 references
    public static string GroupNumbers(List<int> nums)
    {
        Dictionary<string, List<int>> grouped = nums// List<int>
            .GroupBy(n:int => n % 2 == 0 ? "Even" : "Odd")// IEnumerable<IGrouping<…>>
            .ToDictionary(g:IGrouping<string,int> => g.Key, g:IGrouping<string,int> => g.ToLi

        StringBuilder sb = new();
        foreach (KeyValuePair<string, List<int>> group in grouped)
        {
            sb.AppendLine($"{group.Key} numbers: {string.Join(", ", group.Value)}");
        }

        return sb.ToString().Trim();
    }
}
```

You are given a **test file `GroupingTests.cs`** which contains **5 tests**. **One** of them has been **finished partially**, and **four** are **empty** for you to finish:

```
public class GroupingTests
{
    [Test]
    0 references
    public void Test_GroupNumbers_WithEmptyList_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_GroupNumbers_WithEvenAndOddNumbers_ShouldReturnGroupedString()...

    [Test]
    0 references
    public void Test_GroupNumbers_WithOnlyEvenNumbers_ShouldReturnGroupedString()...

    [Test]
    0 references
    public void Test_GroupNumbers_WithOnlyOddNumbers_ShouldReturnGroupedString()...

    [Test]
    0 references
    public void Test_GroupNumbers_WithNegativeNumbers_ShouldReturnGroupedString()...
}
```

When you are ready make sure your **tests run:**



4. Unit Test: Odd Occurrences

Test a given method which takes in an **array of strings** and finds which **words appear** an **odd number** of times.

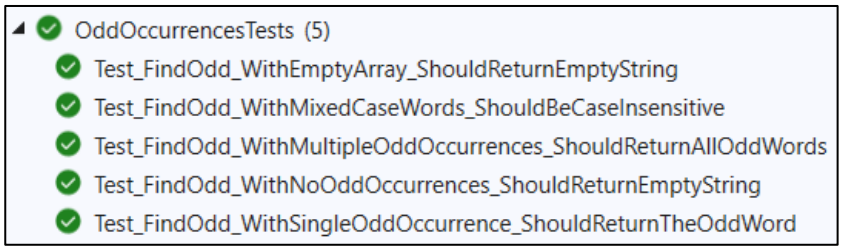
The method is found in the **`OddOccurrences.cs`** file:

```
public class OddOccurrences
{
    1 reference | 0/1 passing
    public static string FindOdd(string[] input)
    {
        Dictionary<string, int> oddWords = new();

        foreach (string word in input)
        {
            string wordLower = word.ToLower();

            oddWords.TryAdd(wordLower, 0);
            oddWords[wordLower]++;
        }

        StringBuilder sb = new();
        foreach (KeyValuePair<string, int> word in oddWords)
        {
            if (word.Value % 2 != 0)
            {
                sb.Append($"{word.Key} ");
            }
        }

        return sb.ToString().Trim();
    }
}
```

You are given a **test file `OddOccurencesTests.cs`** which contains **5 tests**. **One** of them has been **finished partially**, and **four** are **empty** for you to finish:

```
public class OddOccurrencesTests
{
    [Test]
    0 references
    public void Test_FindOdd_WithEmptyArray_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_FindOdd_WithNoOddOccurrences_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_FindOdd_WithSingleOddOccurrence_ShouldReturnTheOddWord()...

    [Test]
    0 references
    public void Test_FindOdd_WithMultipleOddOccurrences_ShouldReturnAllOddWords()...

    [Test]
    0 references
    public void Test_FindOdd_WithMixedCaseWords_ShouldBeCaseInsensitive()...
}
```

When you are ready make sure your **tests run:**



5. Unit Test: Miner

Test a given method which takes in **N number of strings** in the form of:

"**`{mineral} {quantity}`**"

Then it counts the **total quantity of a given mineral** and returns a string showing that.

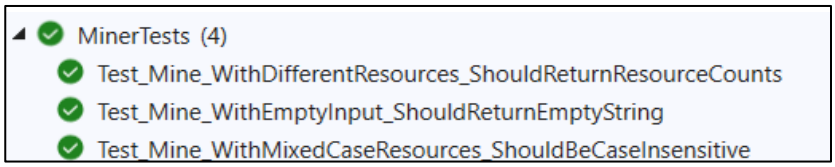
The method is found in the **`Miner.cs`** file:

```
public class Miner
{
    4 references | 0/4 passing
    public static string Mine(params string[] input)
    {
        Dictionary<string, int> resources = new();

        foreach (string s in input)
        {
            string[] split = s.Split();

            resources.TryAdd(split[0].ToLower(), 0);
            resources[split[0].ToLower()] += int.Parse(split[1]);
        }

        StringBuilder sb = new();
        foreach (KeyValuePair<string, int> pair in resources)
        {
            sb.AppendLine($"{pair.Key} -> {pair.Value}");
        }

        return sb.ToString().Trim();
    }
}
```

You are given a **test file `MinerTests.cs`** which contains **4 tests**. **One** of them has been **finished partially**, and **three** are **empty** for you to finish:

```
0 references
public class MinerTests
{
    [Test]
    | 0 references
    public void Test_Mine_WithEmptyInput_ShouldReturnEmptyString()...

    [Test]
    | 0 references
    public void Test_Mine_WithMixedCaseResources_ShouldBeCaseInsensitive()...

    [Test]
    | 0 references
    public void Test_Mine_WithDifferentResources_ShouldReturnResourceCounts()...
}
```

When you are ready make sure your **tests run:**



6. Unit Test: Orders

Test a given method which takes in **N number of strings** in the form of:

"**`{product} {price} {quantity}`**"

It saves **each product** and **quantity** and **updates** the **price** each time it **changes**, and finally **calculates** the **total price** for each **product**.

The method is found in the **`Orders.cs`** file:

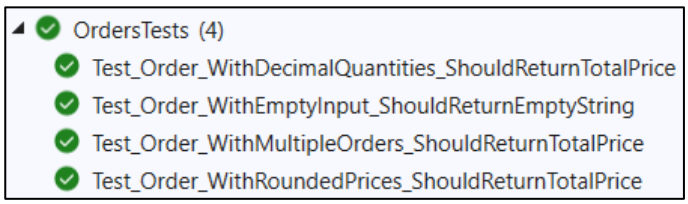
```
public class Orders
{
    4 references | 0/4 passing
    public static string Order(params string[] input)
    {
        Dictionary<string, decimal[]> products = new();

        foreach (string s in input)
        {
            string[] data = s.Split();

            string product = data[0];
            decimal price = decimal.Parse(data[1]);
            decimal quantity = decimal.Parse(data[2]);

            products.TryAdd(product, new[] { (decimal)0.0, (decimal)0.0 });
            products[product][1] += quantity;
            products[product][0] = price;
        }
```

```
        StringBuilder sb = new();
        foreach (KeyValuePair<string, decimal[]> pair in products)
        {
            decimal totalPrice = pair.Value[1] * pair.Value[0];
            sb.AppendLine($"{pair.Key} -> {totalPrice:f2}");
        }

        return sb.ToString().Trim();
    }
}
```

You are given a **test file `OrdersTests.cs`** which contains **4 tests**. **One** of them has been **finished partially**, and **three** are **empty** for you to finish:

```
public class OrdersTests
{
    [Test]
    0 references
    public void Test_Order_WithEmptyInput_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_Order_WithMultipleOrders_ShouldReturnTotalPrice()...

    [Test]
    0 references
    public void Test_Order_WithRoundedPrices_ShouldReturnTotalPrice()...

    [Test]
    0 references
    public void Test_Order_WithDecimalQuantities_ShouldReturnTotalPrice()...
}
```

When you are ready make sure your **tests run:**



7. Unit Test: Plants

Test a given method which takes in an **array of strings** which saves and groups plants based on their number of letters, the shortest named plants will grow the **fastest**.

The method is found in the **`Plants.cs`** file:

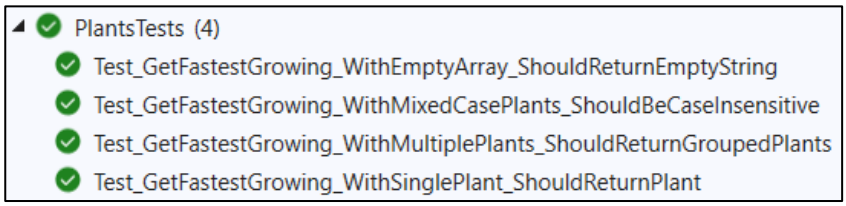
```
public class Plants
{
    4 references | 0/4 passing
    public static string GetFastestGrowing(string[] plants)
    {
        Dictionary<int, List<string>> groupedPlants = new();

        foreach (string plant in plants)
        {
            int length = plant.Length;
            groupedPlants.TryAdd(length, new List<string>());
            groupedPlants[length].Add(plant);
        }
```

```
        StringBuilder sb = new();
        foreach (KeyValuePair<int, List<string>> kvp in groupedPlants.OrderBy(kv :KeyValuePair<int,List<…>> => kv.Key))
        {
            sb.AppendLine($"Plants with {kvp.Key} letters:");
            foreach (string plant in kvp.Value)
            {
                sb.AppendLine(plant);
            }
        }

        return sb.ToString().Trim();
    }
}
```

You are given a **test file `PlantsTests.cs`** which contains **4 tests**. **One** of them has been **finished partially**, and **three** are **empty** for you to finish:

```
public class PlantsTests
{
    [Test]
    0 references
    public void Test_GetFastestGrowing_WithEmptyArray_ShouldReturnEmptyString()...

    [Test]
    0 references
    public void Test_GetFastestGrowing_WithSinglePlant_ShouldReturnPlant()...

    [Test]
    0 references
    public void Test_GetFastestGrowing_WithMultiplePlants_ShouldReturnGroupedPlants()...

    [Test]
    0 references
    public void Test_GetFastestGrowing_WithMixedCasePlants_ShouldBeCaseInsensitive()...
}
```

When you are ready make sure your **tests run:**



At the end make sure all tests pass: