Exam Preparation I

Submit your solutions here: https://judge.softuni.org/Contests/Practice/Index/4512

1. Unit Test: String Rotator

Test a given method which takes in **a string representing the input and an integer representing positions.** The method takes the string and **rotates it by the positions**.

Example: string "person" with 3 positions becomes "sonper"

Examples

Argument	Returned string value
hello 0	hello
abcdef 2	efabcd
12345 -2	45123
xyz 5	yzx

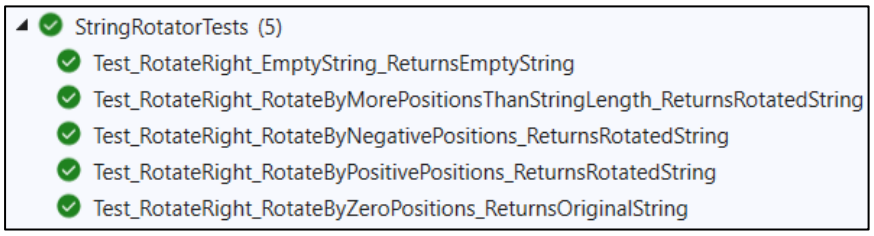
The method is found in the **`StringRotator.cs`** file:

```
public class StringRotator
{
    5 references
    public static string RotateRight(string input, int positions)
    {
        if (string.IsNullOrEmpty(input))
        {
            return input;
        }

        int length = input.Length;
        positions = Math.Abs(positions);
        positions %= length;

        return input.Substring(length - positions)
               + input.Substring(startIndex: 0, length: length - positions);
    }
}
```

You are given a **test file `StringRotatorTests.cs`** containing **5 empty tests**. Implement all tests:

```
public class StringRotatorTests
{
    [Test]
    0 references
    public void Test_RotateRight_EmptyString_ReturnsEmptyString()...

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

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

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

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

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



IMPORTANT: DO NOT REMOVE OR CHANGE ANY NAMESPACES AND USINGS.

2. Unit Test: Dictionary Intersection

Test a given method which takes in **2 `<string, int>` dictionaries** and **finds the intersection of the two**.

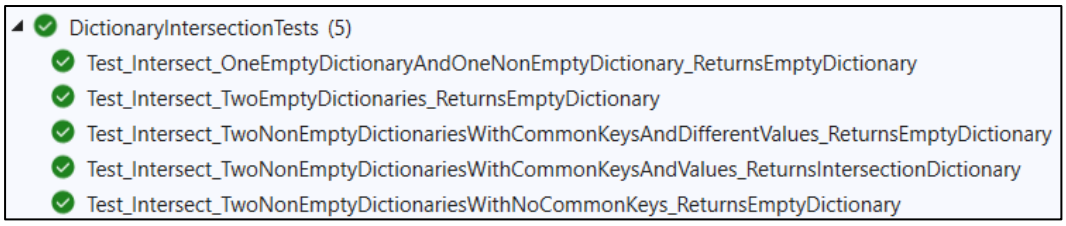
The method is found in the **`DictionaryIntersection.cs`** file:

```
public class DictionaryIntersection
{
    5 references
    public static Dictionary<string, int> Intersect(
        Dictionary<string, int> dict1,
        Dictionary<string, int> dict2)
    {
        Dictionary<string, int> intersection = new();

        foreach (KeyValuePair<string, int> kvp in dict1)
        {
            if (dict2.ContainsKey(kvp.Key) && dict2[kvp.Key] == kvp.Value)
            {
                intersection[kvp.Key] = kvp.Value;
            }
        }

        return intersection;
    }
}
```

You are given a **test file `DictionaryIntersectionTests.cs`** containing **5 empty tests**. Implement all tests:

```
public class DictionaryIntersectionTests
{
    [Test]
    public void Test_Intersect_TwoEmptyDictionaries_ReturnsEmptyDictionary()...

    [Test]
    public void Test_Intersect_OneEmptyDictionaryAndOneNonEmptyDictionary_ReturnsEmptyDictionary()...

    [Test]
    public void Test_Intersect_TwoNonEmptyDictionariesWithNoCommonKeys_ReturnsEmptyDictionary()...

    [Test]
    public void Test_Intersect_TwoNonEmptyDictionariesWithCommonKeysAndValues_ReturnsIntersectionDictionary()...

    [Test]
    public void Test_Intersect_TwoNonEmptyDictionariesWithCommonKeysAndDifferentValues_ReturnsEmptyDictionary()...
}
```

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



IMPORTANT: DO NOT REMOVE OR CHANGE ANY NAMESPACES AND USINGS.

3. Unit Test: Product

You are given a **folder of 2 classes - `Product`** and **`ProductInventory`**. The **`Product` class** is just a helper class:

```
public class Product
{
    2 references
    public string Name { get; init; } = null!;

    3 references
    public double Price { get; init; }

    3 references
    public int Quantity { get; init; }
}
```

The **`ProductInventory`** class holds a **list of products** and **methods** for **using the list** that you will **test**:

```
public class ProductInventory
{
    private readonly List<Product> _products = new();

    5 references
    public void AddProduct(string name, double price, int quantity)
    {
        Product newProduct = new()
        {
            Name = name,
            Price = price,
            Quantity = quantity,
        };

        this._products.Add(newProduct);
    }
```

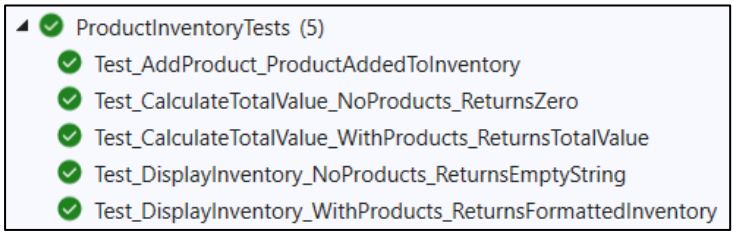
```
    public string DisplayInventory()
    {
        StringBuilder sb = new();

        sb.AppendLine("Product Inventory:");
        foreach (Product product in this._products)
        {
            sb.AppendLine($"{product.Name} - Price: ${product.Price:f2} - " +
                          $"Quantity: {product.Quantity}");
        }

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

```
    public double CalculateTotalValue()
    {
        return this._products.Sum(product => product.Price * product.Quantity);
    }
}
```

You will need to use the test file **`ProductInventoryTests.cs`**, inside they are **5 empty tests with a setup method:**

```
public class ProductInventoryTests
{
    private ProductInventory _inventory = null!;

    [SetUp]
    0 references
    public void SetUp()...

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

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

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

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

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

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



IMPORTANT: DO NOT REMOVE OR CHANGE ANY NAMESPACES AND USINGS.