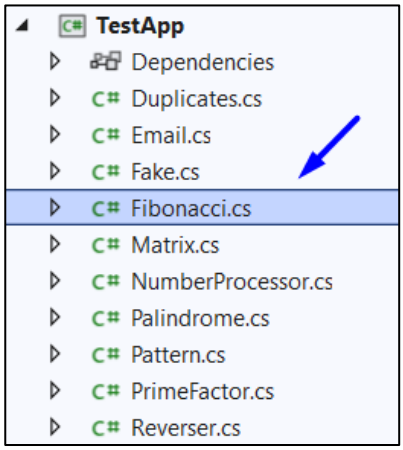
Exercises: Unit Testing Methods, Arrays and Lists

Submit your solutions here: https://judge.softuni.org/Contests/4682/Unit-Testing-Methods-Arrays-and-Lists-Exercise

1. Unit Test Method: Fibonacci

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



The method takes in an **integer** , and calculates the **Fibonacci number**:

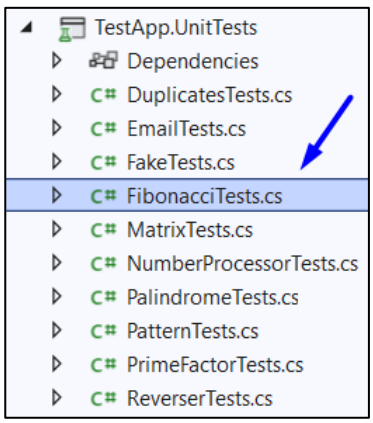
```
public class Fibonacci
{
    5 references
    public static int CalculateFibonacci(int n)
    {



        if (n <= 1)
        {
            return n;
        }

        return CalculateFibonacci(n: n - 1) + CalculateFibonacci(n: n - 2);
    }
}
```

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



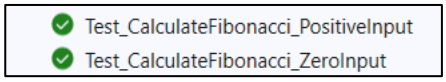
```
public class FibonacciTests
{



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

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

The first test if **finished** so you have a **reference**, 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 Method: Email

Test a given method which takes in a **string representing an email address** and finds if the given email is indeed **a valid email or not**.

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

```
public class Email
{
    3 references
    public static bool IsValidEmail(string email)
    {
        if (string.IsNullOrWhiteSpace(email))
        {
            return false;
        }

        return MailAddress.TryCreate(email, out _);
    }
}
```

You are given again a **test file `EmailTests.cs`** which contains **3 tests**. One of them has been **finished partially**, and **two** are **empty** for you to finish:

```
public class EmailTests
{
    [Test]
    0 references
    public void Test_IsValidEmail_ValidEmail()...

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

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

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

3. Unit Test Method: Prime Factor

Test a given method which takes in a **long number** and finds the largest prime factor of the given number.

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

```
public class PrimeFactor
{
    3 references
    public static long FindLargestPrimeFactor(long number)
    {



        long largestFactor = 1;
        long divisor = 2;

        while (number > 1)
        {
            if (number % divisor == 0)
            {
                largestFactor = divisor;
                number /= divisor;
                continue;
            }

            divisor++;
        }

        return largestFactor;
    }
}
```

In the **test file `PrimeFactorTests.cs`** you are given **3 empty tests**. Finish them and run them:

```
public class PrimeFactorTests
{



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

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

4. Unit Test Array: Reverser

Test a given method which takes in an **array of strings** and **reverses** each string in the array.

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

```
public class Reverser
{
    4 references
    public static string[] ReverseStrings(string[] arr)
    {
        return arr.Select(s:string => new string(s.Reverse().ToArray())).ToArray();
    }
}
```

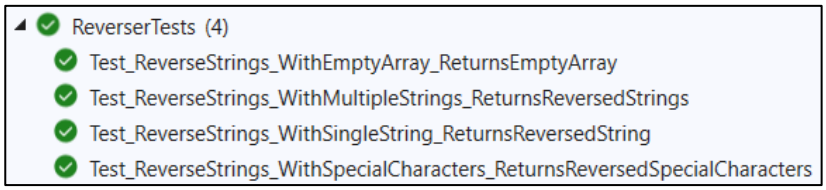
In the **test file `ReverserTests.cs`** you are given **1 finished test**, **1 partially finished test**, and **2 empty tests**. Finish them and run them:

```
public class ReverserTests
{
    [Test]
    0 references
    public void Test_ReverseStrings_WithEmptyArray_ReturnsEmptyArray()...

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

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

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



5. Unit Test Array: Duplicates

Test a given method which takes in an **array of integers** and **removes all duplicate numbers**.

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

```
public class Duplicates
{
    4 references
    public static int[] RemoveDuplicates(int[] numbers)
    {
        HashSet<int> uniqueNumbers = new();

        foreach (int number in numbers)
        {
            uniqueNumbers.Add(number);
        }

        return uniqueNumbers.ToArray();
    }
}
```

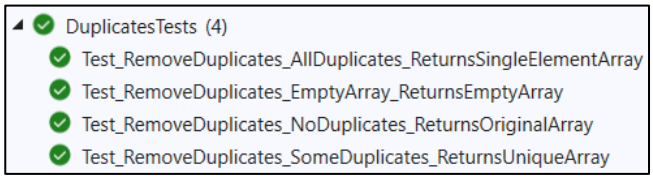
In the **test file `DuplicatesTests.cs`** you are given **2 partially finished test**, and **2 empty tests**. Finish them and run them:

```
public class DuplicatesTests
{
    [Test]
    public void Test_RemoveDuplicates_EmptyArray_ReturnsEmptyArray()...

    [Test]
    public void Test_RemoveDuplicates_NoDuplicates_ReturnsOriginalArray()...

    [Test]
    public void Test_RemoveDuplicates_SomeDuplicates_ReturnsUniqueArray()...

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



6. Unit Test Array: Fake

Test a given method which takes in an **array of characters** and **removes all characters which are numbers and not letters.**

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

```
public class Fake
{
    public static char[] RemoveStringNumbers(char[]? arr)
    {



        return arr.Where(c:char => !char.IsDigit(c)).ToArray();
    }
}
```

Notice the method throws **`ArgumentException`** if the array is **null**.

In the **test file `FakeTests.cs`** you are given **3 empty tests**:

```
public class FakeTests
{



    [Test]
    public void Test_RemoveStringNumbers_RemovesDigitsFromCharArray()...

    [Test]
    public void Test_RemoveStringNumbers_NoDigitsInInput_ReturnsSameArray()...

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

7. Unit Test Array: Pattern

Test a given method which takes in an **array of integers** that **removes all duplicates** and **sorts** the list in a zig-zag pattern.

Example: If we have an array **`1 2 1 3 4 10 12 15`** the sorted array would be: **`1 15 2 12 3 10 4`**.

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

```
public class Pattern
{
    4 references
    public static int[] SortInPattern(int[]? arr)
    {
        Array.Sort(arr);

        int[] distinctList = arr.Distinct().ToArray();

        int[] result = new int[distinctList.Length];
        int left = 0;
        int right = distinctList.Length - 1;
        bool isLeftTurn = true;

        for (int i = 0; i < distinctList.Length; i++)
        {
            if (isLeftTurn)
            {
                result[i] = distinctList[left];
                left++;
            }
            else
            {
                result[i] = distinctList[right];
                right--;
            }

            isLeftTurn = !isLeftTurn;
        }

        return result;
    }
}
```

In the **test file `PatternTests.cs`** you are given **tests**:

```
public class PatternTests
{

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

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

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

8. Unit Test List: Number Processor

Test a given method which takes in a **list of integers** and adds each number in a new list following these principles:

- If the number is **even**, it **squares** it.
- If the number is **odd**, it **square roots** it**.**

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

```
public class NumberProcessor
{
    5 references
    public static List<double> ProcessNumbers(List<int> numbers)
    {
        List<double> result = new();

        foreach (int number in numbers)
        {
            if (number % 2 == 0)
            {
                result.Add(item: Math.Pow(number, 2));
            }
            else
            {
                result.Add(item: Math.Sqrt(number));
            }
        }

        return result;
    }
}
```

In the **test file `NumberProcessorTest.cs`** you are given **tests**. Finish them and run them:

```
public class NumberProcessorTests
{
    [Test]
    0 references
    public void Test_ProcessNumbers_SquareEvenNumbers()...

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

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

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


}
```

9. Unit Test List: Palindrome

Test a given method which takes in a **list of strings** and checks if every word in the array is a **palindrome**.

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

```
public class Palindrome
{
    5 references
    public static bool IsPalindrome(List<string> words)
    {
        return words// List<string>
            .Select(s:string => s.ToLower())// IEnumerable<string>
            .All(word:string => word.SequenceEqual(word.Reverse()));
    }
}
```

In the **test file `PalindromeTests.cs`** you are given **2 partially finished test**, and **3 empty tests**. Finish them and run them:

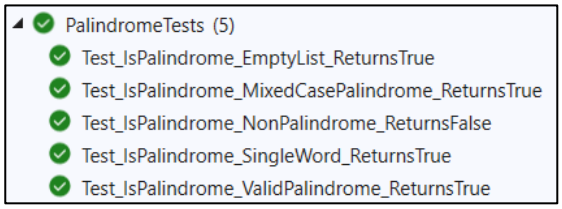
```
public class PalindromeTests
{
    [Test]
    0 references
    public void Test_IsPalindrome_ValidPalindrome_ReturnsTrue()...

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

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

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

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



10. * Unit Test List: Matrix

Test a given method which takes in **2 lists of list of integers (matrix)** that performs **matrix addition** on them.

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

```
public class Matrix
{
    5 references
    public static List<List<int>> MatrixAddition(
        List<List<int>> matrixA,
        List<List<int>> matrixB)
    {
        if (matrixA.Count == 0 || matrixB.Count == 0)
        {
            return new List<List<int>>();
        }

        if (matrixA.Count != matrixB.Count || matrixA[0].Count != matrixB[0].Count)
        {
            throw new ArgumentException(
                message: "Matrices must have the same dimensions for addition.");
        }

        List<List<int>> result = new();
```

```
        for (int i = 0; i < matrixA.Count; i++)
        {
            List<int> row = new();

            for (int j = 0; j < matrixA[0].Count; j++)
            {
                row.Add(item: matrixA[i][j] + matrixB[i][j]);
            }

            result.Add(row);
        }

        return result;
    }
}
```

Notice the method throws **`ArgumentException`** if the matrices are **not the same length**.

In the **test file `MatrixTests.cs`** you are given **2 partially finished test**, and **3 empty tests**:

```
public class MatrixTests
{
    [Test]
    0 references
    public void Test_MatrixAddition_ValidInput_ReturnsCorrectResult()...

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

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

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

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

Check the tests run successfully: