Workshop

Custom Unit Testing Framework

Overview

In this workshop, you need to build a custom unit testing framework on your own, following the basic principles of OOP. Use it to write tests for the provided class.

Setup

You are provided with a **skeleton**, which contains the following items:

- **`CustomUnitTesting`** project – it contains the following folders:
    - **Asserts**
    - **Attributes**
    - **Exceptions**
    - **TestRunner**
    - **Utilities**
- **`CustomUnitTesting.Tests`** project – it contains a public class **`MyTestClass`**
- **`MyProgram`** project – it contains a **`StartUp`** class with a written logic inside it. Its purpose is to give you info about the passed and failed tests.

Structure

Problem 1. CustomUnitTestingFramework

Asserts

In the **Asserts** folder in the **`CustomUnitTesting`** project create the following classes:

Assert

It's a public static class and it needs to hold the Assert methods, which you are going to use when you write tests later.

Write **`public static void AreEqual`** method, which accepts parameters: object a, object b. It should compare if two objects are equal. Here is a logic you can use for it:

```
if (!a.Equals(b))
{
    throw new TestException();
}
```

You are going to create the **`TestException`** class later.

Write **`public static void AreNotEqual`** method, which checks if two objects are different. Again, it accepts the same parameters as the previous method. You can use the following logic for it:

```
if (a.Equals(b))
{
    throw new TestException();
}
```

Write `public static void Throws<T>(Func<bool> condition)` method, which makes sure that a certain condition throws an exception. You can use the following logic:

```
public static void Throws<T>(Func<bool> condition)
  where T : Exception
{
    try
    {
        condition.Invoke();
    }
    catch (T)
    {
    }
    catch
    {
        throw new TestException();
    }
}
```

Attributes

In the **Attributes** folder in the **`CustomUnitTesting`** project create the following classes:

TestClassAttribute

This is a public class, which inherits **`Attribute`**

TestMethodAttribute

This is also a public class, which inherits **`Attribute`**

Exceptions

In the **Exceptions** folder in the **`CustomUnitTesting`** project create the following class:

TestException

A public class, which inherits the **`Exception`** class.

TestRunner

In the **TestRunner** folder in the **`CustomUnitTesting`** project create the following class:

TestRunner

A public class, which must implement the given interface in the **Contracts** subfolder.

The **`TestRunner`** class holds the functionality that runs the tests. The class must hold a `private readonly ICollection<string> resultInfo`. Don't forget to initialize the collection in the constructor. Inside the Run method, you need to get all the classes, which have the **`TestClassAttribute.`** Then you need to run the tests for all the methods inside those classes that have the **`TestMethodAttribute`**. You can use the following logic:

```
var testClasses = Assembly
    .LoadFrom(assemblyPath)
    .GetTypes()
    .Where(ti => ti.HasAttribute<TestClassAttribute>())
    .ToList();

foreach (var testClass in testClasses)
{
    var testMethods = testClass
        .GetMethods()
        .Where(mi => mi.HasAttribute<TestMethodAttribute>())
        .ToList();

    var testClassInstance = Activator.CreateInstance(testClass);

    foreach (var methodInfo in testMethods)
    {
        try
        {
            methodInfo.Invoke(testClassInstance, null);

            resultInfo.Add($"Method: {methodInfo.Name} - passed!");
        }
        catch (TestException te)
        {
            resultInfo.Add($"Method: {methodInfo.Name} - failed!");
        }
        catch
        {
            resultInfo.Add($"Method: {methodInfo.Name} - unexpected error occured!");
        }
    }
}
```

The method must return the **`resultInfo`** collection.

Utilities

In the **Utilities** folder in the **`CustomUnitTesting`** project create the following class:

ReflectionHelper

This is a **public static class**, which checks if a given class / method has an attribute. It holds a single method:

```
public static bool HasAttribute<T>(this MemberInfo mi)
            where T : Attribute
```

You can use the following logic for it:

```
var hasAttribute = mi.GetCustomAttribute<T>() != null;
return hasAttribute;
```

Problem 2.   CustomUnitTestingFramework.Tests

In the **CustomUnitTestingFramework.Tests** project, create a class MyTestClass, which has **TestClassAttribute** and all the methods in it are with a **TestMethodAttribute**. You need to have three methods, that use the methods from your custom testing framework. They should look like this:

```
[TestMethod]
0 references
public void ShouldSumValues()
{
    int a = 2;
    int b = 3;

    int actualSum = a + b;
    int expectedSum = 5;

    Assert.AreEqual(actualSum, expectedSum);
}

[TestMethod]
0 references
public void ShouldSumValues2()
{
    int a = 2;
    int b = 3;

    int actualSum = a + b;
    int expectedSum = 6;

    Assert.AreNotEqual(actualSum, expectedSum);
}

[TestMethod]
0 references
public void ShouldSumValues3()
{
    var a = new string[5];

    Assert.Throws<IndexOutOfRangeException>(() => a[12] == "Lalala");
}
```

Problem 3. MyProgram

In the **MyProgram** project you are given a **StartUp** class, with a written logic inside it. Try to run your program and don't forget to set MyProgram as a start up project. You need to receive the following output on the console:

```
Method: ShouldSumValues - passed!
Method: ShouldSumValues2 - passed!
Method: ShouldSumValues3 - passed!
```

Congratulations! Now you can try to right your own test class using your custom framework!