Unit Testing **Non** Public Entities

Cpt S 321

Washington State University

NUnit and access modifiers

- NUnit does not allow us to test non-public methods
- So what do we do?
  - Make all methods public? -> NO
  - Test only public methods? -> NO
  - Solution:
    - For internal methods: We will declare our test project to be a ”friend” project of the project under test. This will give us access to internal methods.
    - private methods: We will use Reflection to be able to look for methods with specific signature. This will give us access to private methods.

Testing internal methods

- A friend assembly is an assembly that can see the internal entities of another assembly
- What we want is to declare our Test project (“TestProject”) as a friend assembly to the project under test (“ProjectUnderTest”)
- How?:
  - **Option 1**: If the project you are testing (“ProjectUnderTest”) contains an AssemblyInfo.cs file (check the project -> Properties -> AssemblyInfo.cs), edit it by adding the following:

    ```
    [assembly: InternalsVisibleTo("TestProject")]
    ```

  - **Option 2 (preferred)**: You can edit the .csproj file of ProjectUnderTest by adding the following:

    ```
    <ItemGroup>
      <AssemblyAttribute Include="System.Runtime.CompilerServices.InternalsVisibleTo">
        <_Parameter1>TestProject</_Parameter1>
      </AssemblyAttribute>
    </ItemGroup>
    ```

  - **Option 3**: (Ugly solution…) Add the same statement in any of the .cs files of the project you are testing (“ProjectUnderTest”). You will need to also add the following using statement:

    ```
    using System.Runtime.CompilerServices;
    ```

- That’s all!

Testing internal methods – let’s try it! (1/2)

1. Create a class **ClassToDemoTestingNonPublic** in our **HelloWorld** project and define an internal method **tripleThis** as follows:

   // Method that takes an integer value and returns the value tripled.
   internal static int tripleThis(int aNumber)

2. If you haven’t done so already, create a NUnit test project **HelloWorldTests**

3. In the assembly info file of **HelloWorld** give access to internal entities that are declared in **HelloWorld** to **HelloWorldTests** (see the different options on the previous slide)

Testing internal methods – let’s try it! (2/2)

4. In **HelloWorldTests**, write test cases to test **tripleThis**

    - What types of test cases do we need?

    - How many test cases do we need?

Testing private methods

- We will use Reflection in our test class **ClassToDemoTestingNonPublicTest**
- We will declare an object of the class under test in **ClassToDemoTestingNonPublicTest**
- We will implement a method to look up a method declared in the class under test by passing the name of the method as a string
- We will invoke the retrieved method and test it as we typically do

Testing private methods – let's try it!

```
class ClassToDemoTestingNonPublic
{
        // …
        private int privateInstanceMethod(int aNumber)
        {
            return  aNumber;
        }
        // …
}
```

Testing private methods – let’s try it!

```
using System.Reflection;

public class ClassToDemoTestingNonPublicTest
{
        private ClassToDemoTestingNonPublic objectUnderTest =
                                        new ClassToDemoTestingNonPublic();
        private MethodInfo GetMethod(string methodName){
                if (string.IsNullOrWhiteSpace(methodName))
                                Assert.Fail("methodName cannot be null or whitespace");

                 var method = this.objectUnderTest.GetType()
                                .GetMethod(methodName,
                                BindingFlags.NonPublic | BindingFlags.Static | BindingFlags.Instance);

                if  (method  ==  null)
                        Assert.Fail(string.Format("{0} method not found", methodName));

                return  method;
        }
```

Testing private methods – let’s try it! (cont.)

```
// Still in ClassToDemoTestingNonPublicTest
public void TestPrivateInstanceMethod() {
    // Retrieve the method that we want to test using reflection
    MethodInfo methodInfo = this.GetMethod("privateInstanceMethod");

    // Test the method by calling the MethodBase.Invoke method
    Assert.AreEqual(5,
      methodInfo.Invoke(objectUnderTest, // the object on which
                                          // we are invoking the method
      new object[] { 5 } // the list of parameters (in our case, just one)
     ));
}
```

Are we done?

- Not really

- What is the problem with our current implementation and how can we fix it?

    - Hint: Check the Type.GetMethod Method

    - Fix it!