UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Atividade 1 - Unit testing private methods in C#

Link do Github: https://github.com/markkyn/Teste_Software_2024_Santos_Marcos

Testes de Software

Marcos Gabriel da Silva Santos

2024

1. Problema Escolhido

Unit testing private methods in C#

Asked 12 years, 6 months ago Modified 1 month ago Viewed 389k times

426

Visual Studio allows unit testing of private methods via an automatically generated accessor class. I have written a test of a private method that compiles successfully, but it fails at runtime. A fairly minimal version of the code and the test is:

```
//in project MyProj
class TypeA
{
    private List<TypeB> myList = new List<TypeB>();

    private class TypeB
    {
        public TypeB()
        {
        }
    }

    public TypeA()
    {
    }

    private void MyFunc()
    {
        //processing of myList that changes state of instance
    }
}

//in project TestMyProj
public void MyFuncTest()
{
    TypeA_Accessor target = new TypeA_Accessor();
    //following line is the one that throws exception
    target.myList.Add(new TypeA_Accessor.TypeB());
    target.MyFunc();

    //check changed state of target
}
```

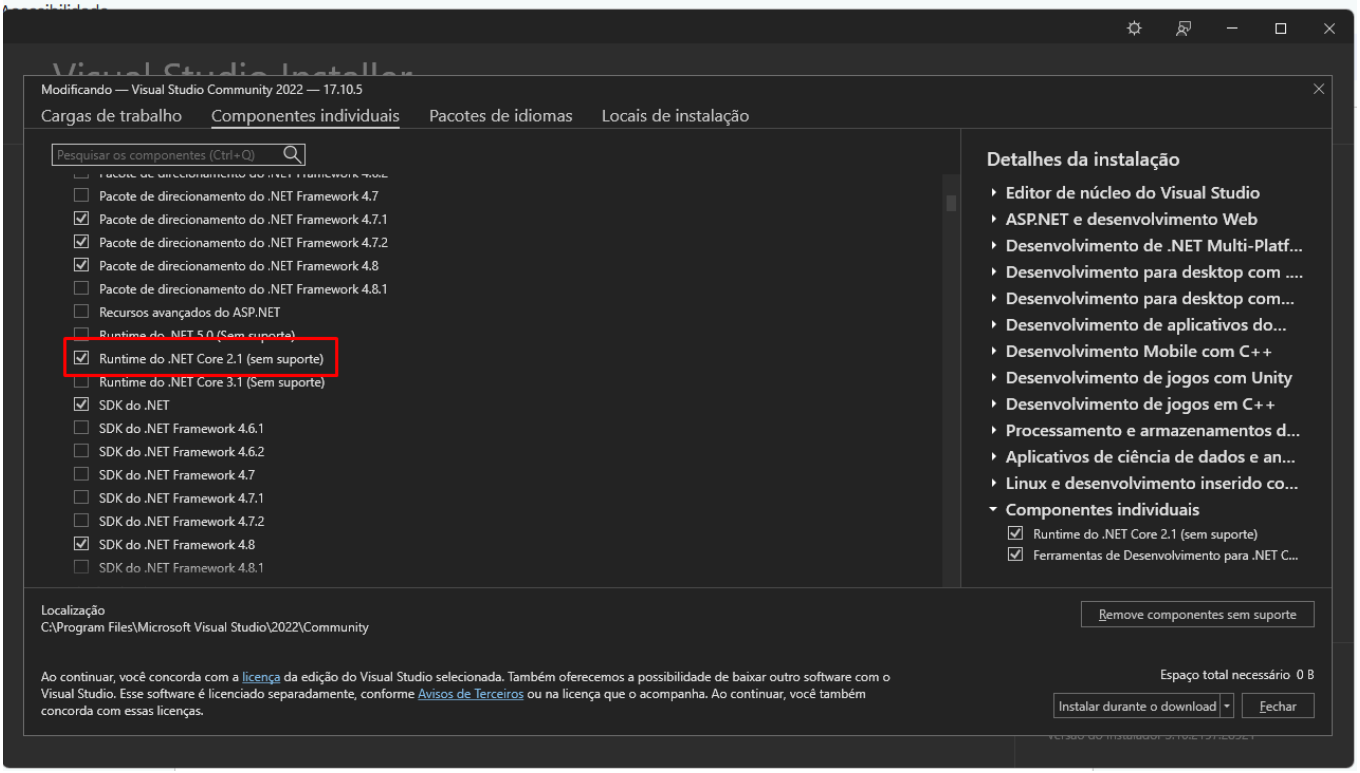
The runtime error is:

```
ot be converted to type 'System.Collections.Generic.List`1[MyProj.TypeA.TypeA+TypeB]'.
```

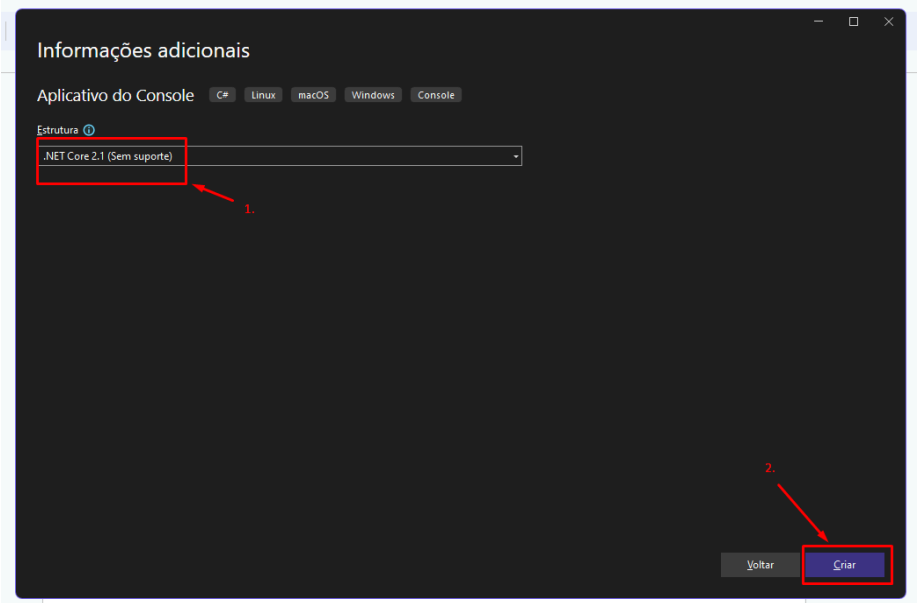
According to intellisense - and hence I guess the compiler - target is of type TypeA_Accessor. But at runtime it is of type TypeA, and hence the list add fails.

Is there any way I can stop this error? Or, perhaps more likely, what other advice do other people have (I predict maybe "don't test private methods" and "don't have unit tests manipulate the state of objects").

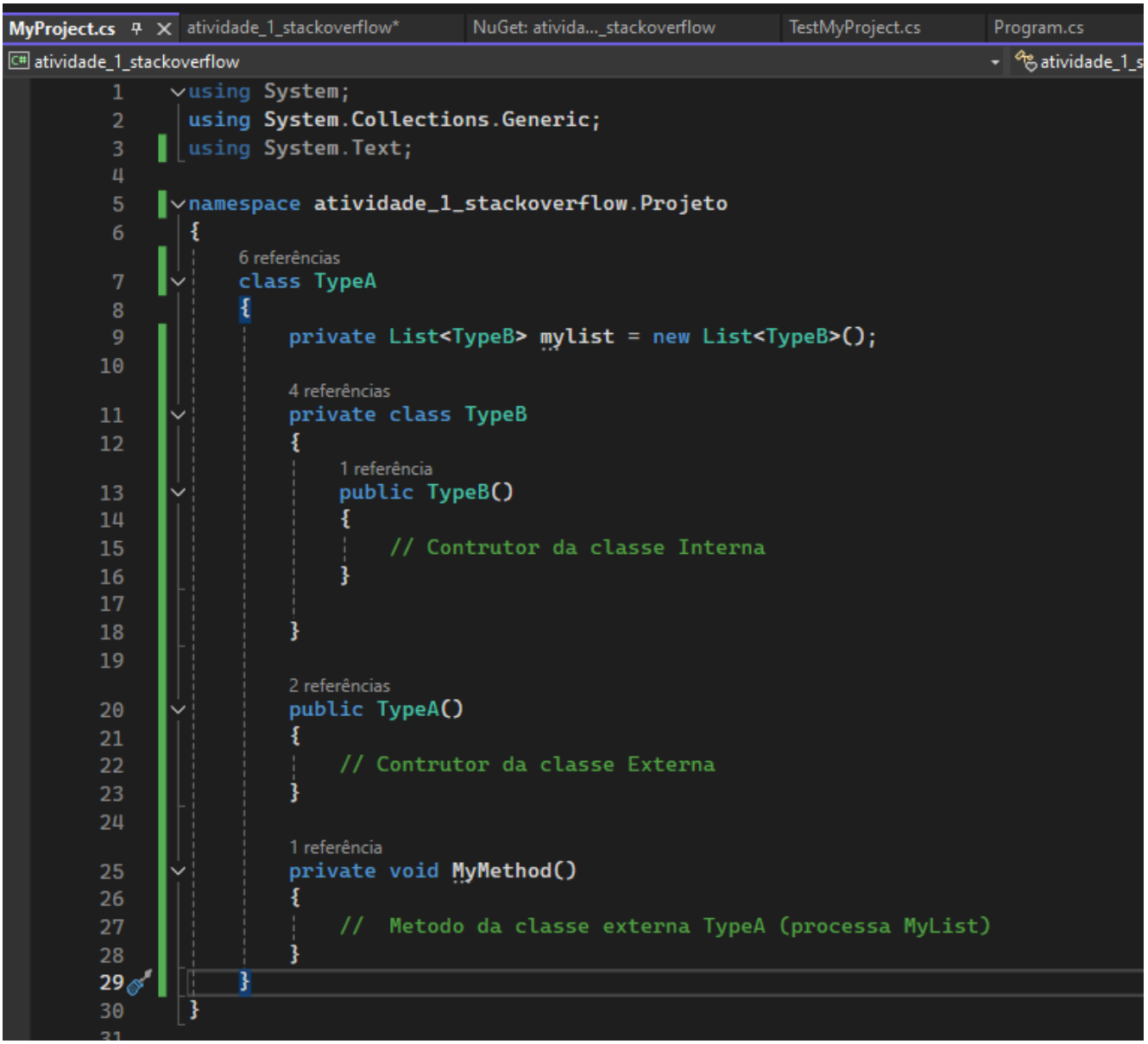
O desenvolvedor que está com problema em testar os métodos e as classes internas por conta do encapsulamento que está em “private”, e em seu projeto ele ainda comenta que o compilador .NET, não impede a compilação, mas o Runtime do .NET gera uma exceção em tempo de execução. Vale ressaltar que a pergunta tem 12 anos e 6 meses, e hoje já estamos na versão 8 do .NET, para isso precisamos instalar a versão mais próxima do contexto da pergunta. Dessa forma, será necessário instalar a versão 2.1 do .NET CORE (Sem suporte), na figura abaixo, utilizando o Visual Studio Installer, basta pesquisar pelo **componente individual “Runtime do .NET Core 2.1 (Sem suporte)”.**



O desenvolvedor disponibilizou um código-exemplo para representar sua dúvida, mesmo com certas inconsistências e está um pouco incompleto , ainda, é possível realizar os testes dessa atividade. Desse modo, vamos criar o projeto .NET com o RunTime instalado:



Com o projeto criado, vamos criar as classes e para tentar replicar o erro:



Agora vamos criar a função de teste que foi definida no problema para testar a classe “TypeA” bem como sua classe interna “TypeB”. Diferentemente do que foi proposto no problema, não foi possível replicar a compilação bem-sucedida por conta do nível de acesso “privado” da classe interna e do método. Isso pode se dar, muito provavelmente, por falta de código disponibilizado pelo dono da pergunta. É possível, ainda, verificar que o dono da pergunta utilizou uma classe não definida “TypeA_Accessor”, o que apoia a argumentação anterior.

Segue a função de teste:

```
namespace atividade_1_stackoverflow
{
    0 referências
    public class Program
    {
        0 referências
        static void Main(string[] args)
        {
            MyFuncTest();
        }
        1 referência
        static void MyFuncTest()
        {
            TypeA_Accessor target = new TypeA_Accessor();

            target.myList.Add(new TypeA.TypeB());

            target.MyMethod();
        }

        0 referências
        public void Solu
```

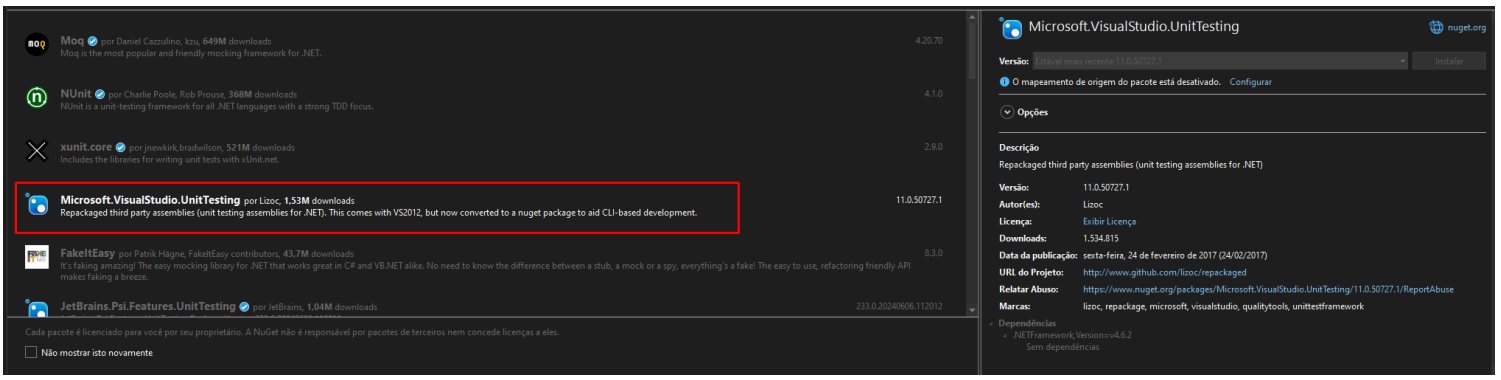
void TypeA.MyMethod()

CS0122: "TypeA.MyMethod()" é inacessível devido ao seu nível de proteção

Mesmo sem conseguir replicar a compilação bem-sucedida é possível analisar a dúvida descrita, visto que ele quer testar os métodos privados da classe “TypeA”.

2. Solução 1

A principal solução , com 830 “up-votes”, utiliza da classe **PrivateObject** para poder testar atributos e métodos privados de uma um objeto. Nesse sentido, a solução ainda utiliza do **Assert.AreEqual** para comparar o valor privado com o valor esperado. Para que possamos utilizar dessa classe, será necessário instalar o pacote **“Microsoft.VisualStudio.TestTools.UnitTesting”** que pode ser encontrado no gerenciador de pacotes NuGet.



Desse modo, vamos replicar a solução.

20 Answers

Sorted by: Highest score (default)

830

If using a version prior to .NET Core 2.0, you can use the `PrivateObject` class:

```
Class target = new Class();
PrivateObject obj = new PrivateObject(target);
var retVal = obj.Invoke("PrivateMethod");
Assert.AreEqual(expectedVal, retVal);
```

`PrivateObject` and `PrivateType` support was removed in .NET Core 2.0.

Share Edit Follow

edited Mar 20, 2023 at 23:36
Andrew D. Bond
1,150 • 1 • 15 • 12

answered Mar 25, 2013 at 4:10
Scuttle
8,397 • 2 • 14 • 10

34 This is the correct answer, now that Microsoft has added PrivateObject. – Zoey Jan 27, 2014 at 11:57

4 Good answer but please note that the PrivateMethod needs to be "protected" in stead of "private". – HerbalMart Mar 13, 2014 at 11:54

27 @HerbalMart: Perhaps I misunderstand you, but if you are suggesting that PrivateObject can only access protected members and not private ones, you are mistaken. – kmote Jun 18, 2014 at 18:37

26 @JeffPearce For static methods you can use "PrivateType pt = new PrivateType(typeof(MyClass));", and then call InvokeStatic on the pt object as you would call Invoke on a private object. – Steve Hibbert May 2, 2017 at 9:45

MyProject.cs | atividade_1_stackoverflow | NuGet: atividade..._stackoverflow | TestMyProject.cs | Program

atividade_1_stackoverflow

```
1   using System;
2   using Microsoft.VisualStudio.TestTools.UnitTesting;
3
4   using atividade_1_stackoverflow.Projeto;
5
6
7   namespace atividade_1_stackoverflow
8   {
        0 referências
9       public class Program
10      {
            0 referências
11          static void Main(string[] args)
12          {
13              Solucao();
14          }
15
            1 referência
16          static void Solucao()
17          {
18              TypeA target = new TypeA();
19              PrivateObject obj = new PrivateObject(target);
20              var retVal = obj.Invoke("MyMethod");
21              Assert.AreEqual(retVal, "Hello World!");
22
23              System.Console.WriteLine("Teste Passado");
24          }
25      }
26  }
27
```

Para podermos testar essa solução, vamos alterar um pouco o método privado da classe “TypeA” para exemplificar o teste unitário.

```
0 referências
private string MyMethod()
{
    //  Metodo da classe externa TypeA (processa MyList)
    return "Hello World!";

}
```

Como não há um acesso direto ao método privado agora é possível compilar o projeto encapsulado, sem problemas:

```
1>D:\Academico\Disciplinas - UFS\Testes de Software\atividade_1_stackoverflow\atividade_
1>atividade_1_stackoverflow -> D:\Academico\Disciplinas - UFS\Testes de Software\atividade
1>Projeto de compilação pronto "atividade_1_stackoverflow.csproj".
========== Compilação: 1 bem-sucedida, 0 com falha, 0 atualizada, 0 ignorada ==========
========== Compilação concluído às 17:46 e levou 03,098 segundos ==========
```

Agora, vamos executar a solução!

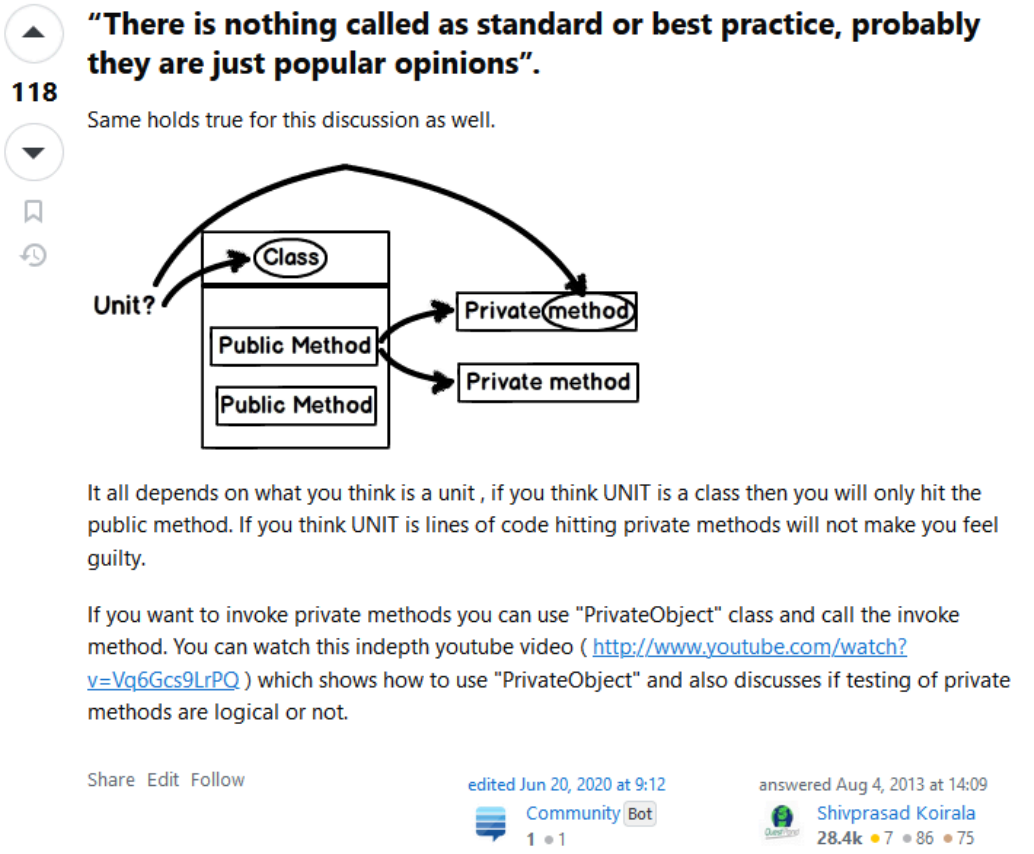
```
Console de Depuração do Mic
Teste Passado

O C:\Program Files\dotnet\dotnet.exe (processo 32752) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas -> Opções -> Depuração -> Fechar o
console automaticamente quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...
```

Com isso, conseguimos testar um método privado dentre de uma objeto do projeto.

3. Outras Soluções

A thread possui várias soluções, algumas mais diretas, outras mais conceituais, e possui , até, opiniões não solicitadas. A segunda resposta a esse post impõe uma discussão acerca do conceito e dos limites de um teste unitário. Ele até comenta o uso de **PrivateObject**, mas não é direto, e eu particularmente discordo do seu ponto de vista em que testes em métodos privados não são triviais, nem lógicos, mas acredito que isso não vem ao caso.



Outras respostas criam soluções mais sofisticadas como o uso da diretiva **“[assembly: InternalsVisibleTo(‘Teste’)]”**. Entretanto, acho que esse método pode trazer muito “lixo semântico” ao código, além que pode ocasionar uma complexidade maior que a própria aplicação.

111

Another thought here is to extend testing to "internal" classes/methods, giving more of a white-box sense of this testing. You can use `InternalsVisibleTo` attribute on the assembly to expose these to separate unit testing modules.

In combination with sealed class you can approach such encapsulation that test method are visible only from unittest assembly your methods. Consider that protected method in sealed class is de facto private.

```
[assembly: InternalsVisibleTo("MyCode.UnitTests")]
namespace MyCode.MyWatch
{
    #pragma warning disable CS0628 //invalid because of InternalsVisibleTo
    public sealed class MyWatch
    {
        Func<DateTime> _getNow = delegate () { return DateTime.Now; };


       //construktor for testing purposes where you "can change DateTime.Now"
       internal protected MyWatch(Func<DateTime> getNow)
       {
           _getNow = getNow;
       }

       public MyWatch()
       {
       }
   }
}
```

And unit test:

```
namespace MyCode.UnitTests
{

[TestMethod]
public void TestminuteChanged()
{
    //watch for traviling in time
    DateTime baseTime = DateTime.Now;
    DateTime nowforTesting = baseTime;
    Func<DateTime> _getNowForTesting = delegate () { return nowforTesting; };

    MyWatch myWatch= new MyWatch(_getNowForTesting );
    nowforTesting = baseTime.AddMinute(1); //skip minute
    //TODO check myWatch
}

[TestMethod]
public void TestStabilityOnFebruary29()
{
    Func<DateTime> _getNowForTesting = delegate () { return new DateTime(2024, 2, 29
    MyWatch myWatch= new MyWatch(_getNowForTesting );
    //component does not crash in overlap year
}
}
```