

Práctica 6.3 Pruebas unitarias con NUnit

Objetivo

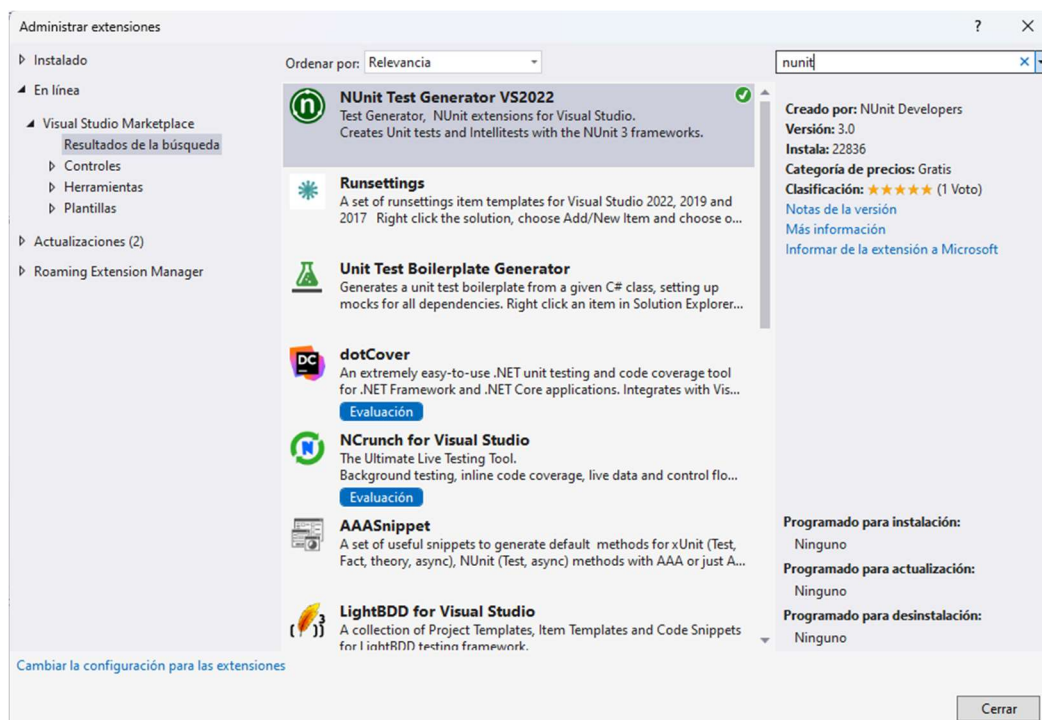
Vamos a utilizar el marco de pruebas para C# **NUnit** y las herramientas que proporciona Visual Studio para automatizar las pruebas unitarias de la clase **Alumno** que se proporciona.

Página del proyecto NUnit: <http://www.nunit.org/>

Información detallada sobre el uso de NUnit: Colabora.NET

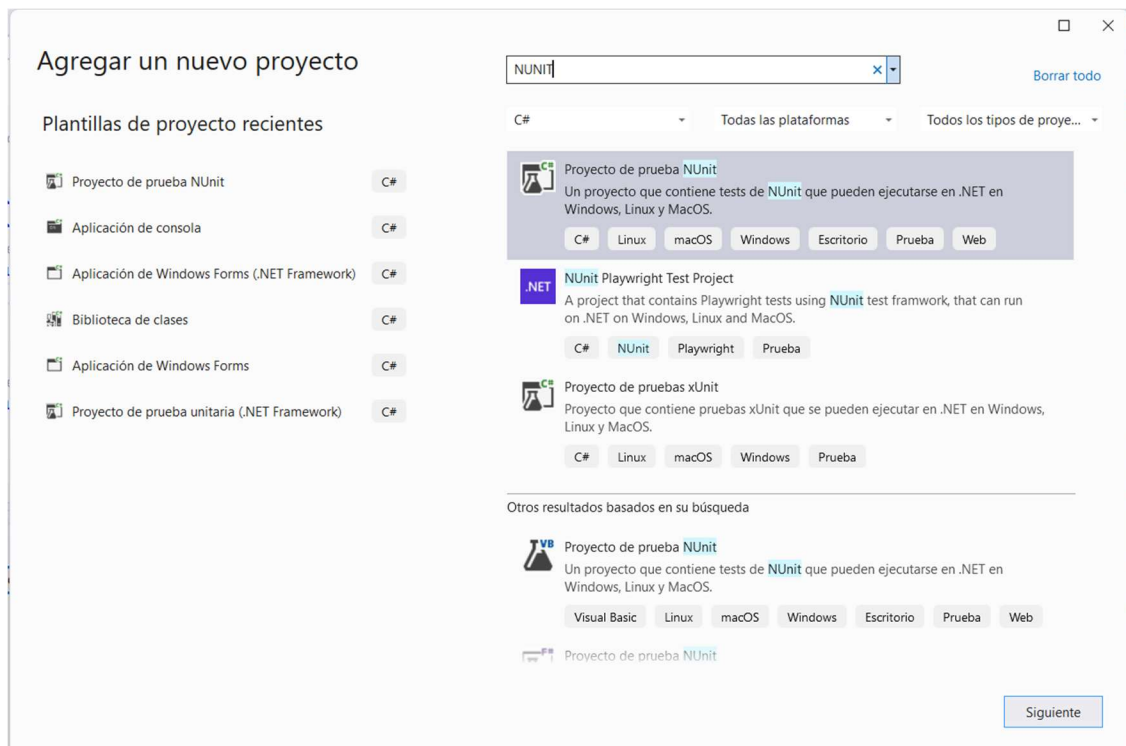
1. Instalación de NUnit

Vamos a instalar la extensión para usar NUnit en Visual Studio. Vamos a Extensiones → Administrar extensiones → En línea y escribimos NUnit en el cuadro de búsqueda. De los resultados obtenidos hay que instalar **Nunit Test Generator VS2022**.



2. Creación de pruebas con NUnit

Se creará un nuevo proyecto de NUnit llamado **TestAlumno**, de la siguiente manera con el botón derecho -> Agregar -> Nuevo Proyecto -> Buscamos el proyecto de prueba NUnit



En el nuevo proyecto TestAlumno en el apartado de Dependencias agregamos la referencia del proyecto inicial para poder usarlo en la práctica de test.

Escribimos el atributo **TestFixture** delante de la declaración de la clase:

```
[TestFixture] clase TestAlumno  
  
{  
  
}
```

A continuación, para cada caso de prueba que tengamos previamente diseñado, creamos un método con el atributo Test, que indicará que el método es un test. Vamos a crear 3 tests para la propiedad Nota, uno válido (4) y 2 no válidos (-4 y 56, recordemos que la nota ha de estar entre 0 y 10).

```
[Test]

public void TestNotaValida()

{

    Alumno alum = new Alumno();

    int nota = 4;

    int esperado = 4;

    alum.Nota = nota;

    Assert.AreEqual(esperado, alum.Nota);

}

[Test]

public void TestNotaNoValida1()

{

    Alumno alum = new Alumno();

    int nota = -4;

    int esperado = 0;

    alum.Nota = nota;

    Assert.AreEqual(esperado, alum.Nota);

}

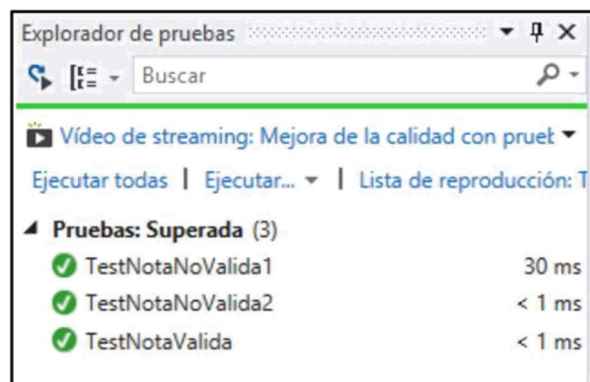
[Test]

public void TestNotaNoValida2()
```

```
{  
  
    Alumno alum = new Alumno();  
  
    int nota = 56;  
  
    int esperado = 0;  
  
    alum.Nota = nota;  
  
    Assert.AreEqual(esperado, alum.Nota);  
  
}
```

Los métodos de prueba deben ser públicos y de tipo **void**. Dentro de cada método creamos un objeto de la clase Alumno y creamos 2 variables: el valor que pasamos y el valor esperado. Asignamos el valor nota a la propiedad Nota del alumno. Por último, usamos el método `Assert.AreEqual(expected, actual)` que compara el **valor esperado** (*expected*) y el **valor devuelto** (*actual*). Si son iguales la prueba estará superada. En el caso válido, el valor esperado es igual al valor devuelto, mientras que en los casos no válidos el valor esperado será 0 (valor por defecto de los campos de tipo *int*).

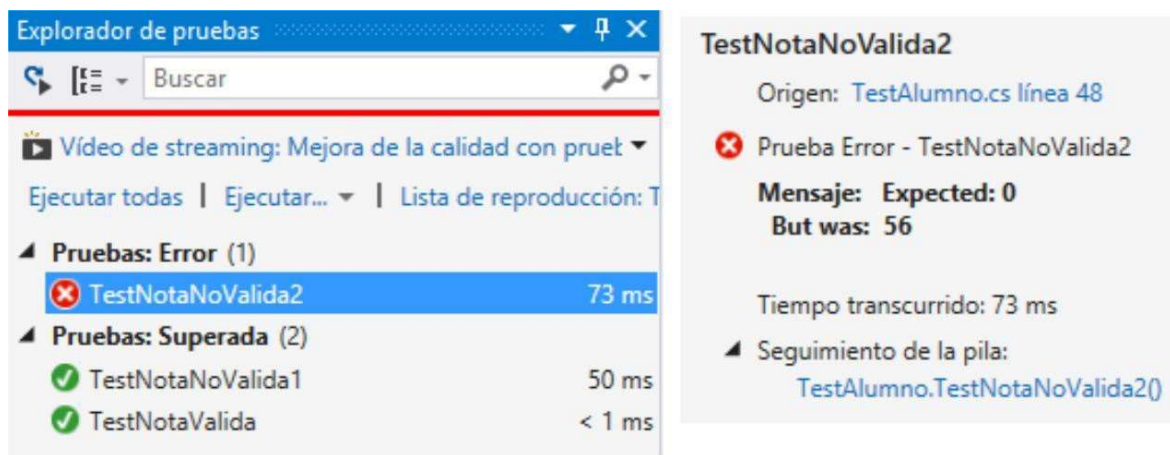
Para ejecutar los métodos de prueba hay que mostrar el explorador de pruebas: *Menú Prueba → Explorador de pruebas*. En esta ventana podemos ejecutar todas las pruebas o sólo las no ejecutadas o no superadas. Hacemos click en Ejecutar todas. El resultado debería ser que las 3 pruebas se han superado:



Vamos a ver qué pasa cuando un test detecta un error en la clase. Cambiamos la condición de entrada de la nota quitando la condición de que la nota sea menor o igual que 10, quedando

```
if (value >= 0)
```

Volvemos a ejecutar todas las pruebas y nos tendría que dar un error en TestNotaValida2:



Explorador de pruebas

Buscar

Vídeo de streaming: Mejora de la calidad con pruet

Ejecutar todas | Ejecutar... | Lista de reproducción: T

Pruebas: Error (1)

- TestNotaNoValida2 73 ms

Pruebas: Superada (2)

- TestNotaNoValida1 50 ms
- TestNotaValida < 1 ms

TestNotaNoValida2

Origen: TestAlumno.cs línea 48

Prueba Error - TestNotaNoValida2

Mensaje: Expected: 0
But was: 56

Tiempo transcurrido: 73 ms

Seguimiento de la pila:
TestAlumno.TestNotaNoValida2()

Con lo que comprobamos que el test detecta un error en el código que hay que corregir.

Corregimos el error dejando la condición anterior y hacemos los tests para las dos propiedades que quedan. Empezamos por **Nombre** y hacemos 2 tests, uno con un valor válido y otro en el que crearemos un objeto Alumno pero no le daremos nombre. El primer test es trivial. Para el segundo debemos comprobar que el valor devuelto es nulo (ya dijimos que el valor por defecto de los campos Sting es **null**), por lo que usaremos el método `Assert.IsNull(object)` que comprueba que el objeto devuelto sea nulo. El código quedaría así:

```
[Test]

public void TestNombre()

{

    Alumno alum = new Alumno();

    string nombre = "Julian";

    string esperado = "Julian";

    alum.Nombre = nombre;

    ClassicAssert.AreEqual(esperado, alum.Nombre);

}

[Test]

public void TestNombreNulo()

{

    Alumno alum = new Alumno();

    ClassicAssert.IsNull(alum.Nombre);

}
```

Por último, nos falta comprobar la propiedad Aprobado. Lo haremos creando un objeto Alumno y pasándole una nota que suponga el aprobado y otra que no. Luego comprobamos el valor devuelto por Aprobado con los métodos `Assert.IsFalse(condition)` y `Assert.IsTrue(condition)` que comprueban si el valor booleano **condition** es **true** o **false**:

```
[Test]

public void TestAprobado()

{

    Alumno alum = new Alumno();

    int nota = 5;

    alum.Nota = nota;

    Assert.IsTrue(alum.Aprobado);

}
```

[Test]

```
public void TestNoAprobado()  
{  
  
    Alumno alum = new Alumno();  
  
    int nota = 4;  
  
    alum.Nota = nota;  
  
    Assert.IsFalse(alum.Aprobado);  
}
```

Si ejecutamos todas las pruebas, deberían superarse todas. En caso contrario tendremos algún error en la clase alumno (¡o en las propias pruebas!).

3. Lista de aserciones más usadas

Método	Función
<code>Assert.AreEqual(expected, actual)</code>	Comprueba que expected y actual tienen el mismo valor.
<code>Assert.AreNotEqual(expected, actual)</code>	Comprueba que expected y actual NO tienen el mismo valor.
<code>Assert.IsTrue(bool condition)</code>	Comprueba que condition es true.
<code>Assert.IsFalse(bool condition)</code>	Comprueba que condition es false.
<code>Assert.IsNull(object anObject)</code>	Comprueba que el objeto anObject es nulo.
<code>Assert.IsNotNull(object anObject)</code>	Comprueba que el objeto anObject NO es nulo.
<code>Assert.AreSame(expected, actual)</code>	Comprueba si los dos argumentos están referenciando al mismo objeto.
<code>Assert.Greater(arg1, arg2)</code>	Comprueba si arg1 es mayor que arg2.

4. Código de la clase TestAlumno

```
using NUnit.Framework;

namespace EjemploAlumno
{
    [TestFixture]

    class TestAlumno
    {
        [Test]

        public void TestNombre()
        {
            Alumno alum = new Alumno();

            string nombre = "Julian";
            string esperado = "Julian";

            alum.Nombre = nombre;

            Assert.AreEqual(esperado, alum.Nombre);
        }

        [Test]

        public void TestNombreNulo()
        {
            Alumno alum = new Alumno();

            Assert.IsNull(alum.Nombre);
        }

        [Test]

        public void TestNotaValida()
        {
            Alumno alum = new Alumno();

            int nota = 4;
            int esperado = 4;
```



```
        alum.Nota = nota;

        Assert.AreEqual(esperado, alum.Nota);

    }

[Test]

public void TestNotaNoValida1()

{

    Alumno alum = new Alumno();

    int nota = -4;

    int esperado = 0;

    alum.Nota = nota;

    Assert.AreEqual(esperado, alum.Nota);

}

[Test]

public void TestNotaNoValida2()

{

    Alumno alum = new Alumno();

    int nota = 56;

    int esperado = 0;

    alum.Nota = nota;

    Assert.AreEqual(esperado, alum.Nota);

}

[Test]

public void TestAprobado()

{

    Alumno alum = new Alumno();

    int nota = 5;

    alum.Nota = nota;

    Assert.IsTrue(alum.Aprobado);

}
```

```
[Test]

public void TestNoAprobado()

{

    Alumno alum = new Alumno();

    int nota = 4;

    alum.Nota = nota;

    Assert.IsFalse(alum.Aprobado);

}

}
```

5. Pruebas parametrizadas

Hasta aquí hemos visto cómo pasar los casos de prueba de uno en uno, pero esto puede ser muy engorroso si tenemos muchos. Por suerte, podemos pasar parámetros a las pruebas para pasar varios valores a cada test. En nuestro ejemplo, supongamos que queremos pasar al método **TestNota-Valida** todas las notas válidas entre el 0 y el 10. Para ello creamos un parámetro llamado *nota* que tomará los valores del 0 al 10 con el atributo **[Range(0,10,1)]** que pasará un rango de valores empezando por el 0 hasta el 10 con incrementos de 1. Tenemos que modificar un poco el cuerpo del método:

```
public void TestNotaValida([Range(0,10,1)] int nota)

{

    Alumno alum = new Alumno();

    alum.Nota = nota;

    Assert.AreEqual(nota, alum.Nota);

}
```

Otros atributos que podemos usar son:

[Values (v1, v2, v3, v4...)] → Pasa los valores indicados.

[Random (a, b, x)] → Pasa x valores en el rango a-b

Esto nos permitirá, por ejemplo, crear un método de test para cada clase de equivalencia y pasarle los casos de prueba que hayamos diseñado, incluyendo valores límite.

6. Cobertura de código

Visual Studio nos proporciona una herramienta para analizar la **cobertura de código** de nuestras pruebas, es decir, el porcentaje total de código ejecutado por las pruebas. Conviene que la cobertura esté lo más cerca posible del 100%, al menos en las clases importantes. En nuestro ejemplo, la clase importante es Alumno.

Para hacer el análisis vamos al menú *Prueba → Analizar cobertura de código → Todas las pruebas*. Esto vuelve a pasar las pruebas y nos muestra los resultados en la ventana *Resultados de la cobertura de código*, en el panel inferior. Si desplegamos los elementos de nuestro proyecto, podemos ver que la clase Alumno tiene una cobertura del 100%.

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% bloques)
▲ {} EjemploAlumno	25	33,78 %	49	66,22 %
▶ 📁 Alumno	0	0,00 %	15	100,00 %
▶ 📁 PruebaManual	25	100,00 %	0	0,00 %
▶ 📁 TestAlumno	0	0,00 %	34	100,00 %

Si además hacemos click en el botón *Mostrar colores en cobertura de código* (en la misma ventana), veremos resaltadas las líneas de código que han sido cubiertas por las pruebas. **El siguiente paso es hacer los test de la clase Alumnos para llegar a una cobertura del 100% de la clase.**

7. Práctica a realizar 1

- 1º Crear cobertura del 100% para la clase Alumnos con los test.
- 2º Añade una opción que antes de salir te muestre todo lo que has insertado por consola. Para ello necesitamos crear el método Longitud dentro de la clase del ArrayList.
- 3º Después de crear este método comprobamos la cobertura de los test y comprobamos si la cobertura de las dos clases son el 100%, si no es así, deberemos crear los test necesarios para que la cobertura sea el 100%

8. Práctica a realizar 2

Hacer una solución con Visual Studio llamada **Practica6.3**. Dentro crear un proyecto con las siguientes clases:

Reserva: clase que implementará el programa del ejercicio 2 de la práctica 6.1. Deberá tener los campos **nif** (de tipo **string**) y **numPlazas** (de tipo **int**) y las propiedades **Nif** y **NumPlazas**, las cuales deberán validar los datos introducidos en sus correspondientes campos.

Para la validación de datos, podremos usar dos alternativas:

- Producir una excepción en caso de que no se haya introducido un valor correcto. Esta es la forma recomendada. Podéis obtener más información sobre el control de excepciones con NUnit en el siguiente enlace: <http://dotnetpattern.com/nunit-assert-examples>. De manera optativa, se puede controlar que la letra del DNI introducida sea realmente correcta (en principio la práctica no requiere de validación de letra).
- No lanzar excepciones, controlando la validez de la introducción de datos

mediante dos propiedades booleanas (**nifValido** y **numPlazasValido**, por ejemplo). -.

TestReserva: clase con las pruebas unitarias para la clase Reserva. Se deberán pasar todos los casos de prueba obtenidos en la práctica 6.1. Además, la cobertura para la clase Reserva deberá ser del 90% o superior. Se recomienda usar pruebas parametrizadas donde sean adecuadas.

Se deberá adjuntar un **documento con los casos de prueba** que se usen. Entregar la carpeta de la solución junto con el documento de casos de prueba (dossier PDF) comprimidos en formato zip.