MEJORANDO EL PROYECTO DE CALCULADORA

Para garantizar la integridad el switch utilizamos un enum para definir las opciones ha elegir ,

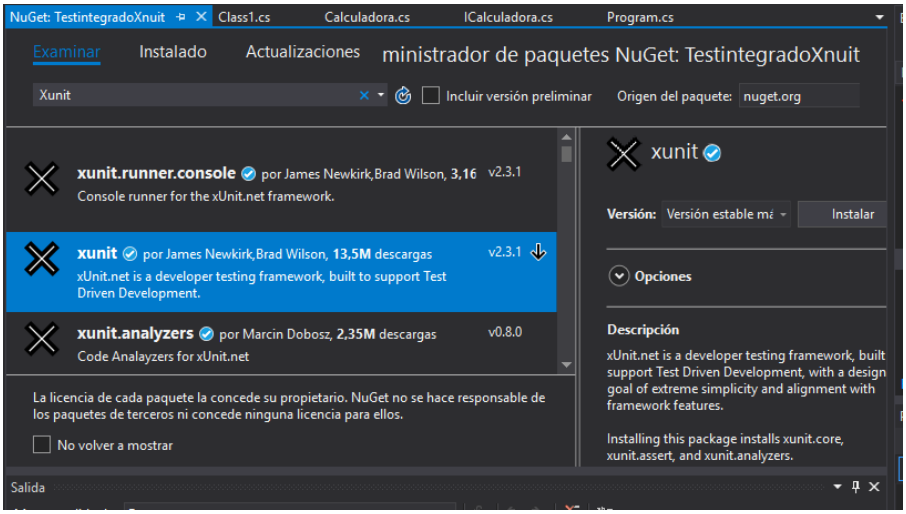
```
5 referencias | Daniel Madrigal, Hace 6 minutos | 1 autor, 1 cambio
public enum Opcion { Suma=1,Resta,Multiplicacion,Division};
0 referencias | Daniel Madrigal, Hace 6 minutos | 1 autor, 1 cambio
```

La manera de utilizar es bastante sencilla basta con castear el int de la opción en la declaración del switch y en los case colocar las opciones definidas en el enum:

```
int num2 = Convert.ToInt32(Console.ReadLine());
//casetamos el int con el enum y asi obtenemos un switch fuertemente tipado
switch ((Opcion)opcion)
{
    case Opcion.Suma:
        resultado = cal.Suma(num1, num2);
        Console.WriteLine("El resultado es " + resultado);
        break;
    case Opcion.Resta:
        resultado = cal.Resta(num1, num2);
        Console.WriteLine("El resultado es " + resultado);
        break;
    case Opcion.Multiplicacion:
        resultado = cal.Multiplicacion(num1, num2);
        Console.WriteLine("El resultado es " + resultado);
        break;
    case Opcion.Division:
        resultado = cal.Division(num1, num2);
        Console.WriteLine("El resultado es " + resultado);
        break;
```

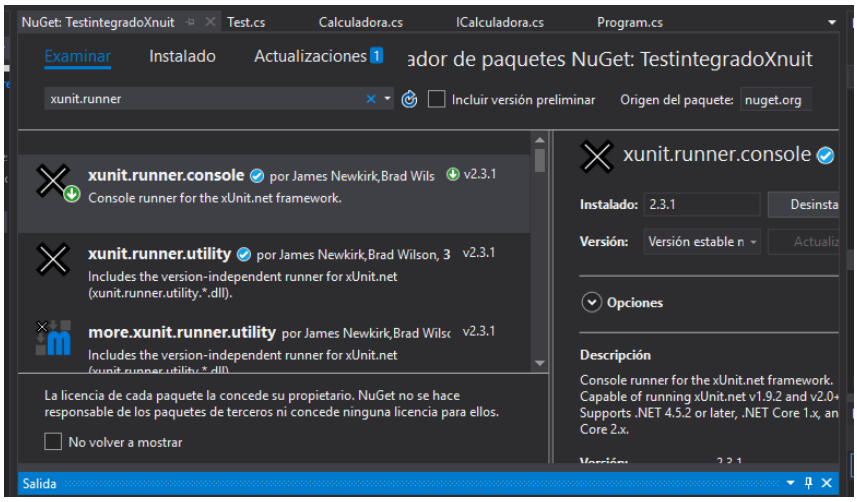
De esta forma tenemos un menú más consistente y con opciones constantes.

Ahora vamos a validar nuestro proyecto utilizando el framework Xunit , para ellos debemos bajarlo del servidor de nuget e incluirlo en nuestro proyecto de biblioteca de clases.



Definiremos los métodos de la clase Calculadora a testear y utilizaremos la parametrización para probar posibles situaciones en nuestros métodos.

Añadimos el paquete nuget para correr los test.



Para implementar los tests parametrizados ya no utilizamos el atributo [Fact]

Empleamos [Theory] y le pasamos los diferentes parámetros a tratar con [InlineData(x,y)]

```
[Theory]
[InlineData(2,2,4)]
[InlineData(4, 6, 10)]
[InlineData(5, 20, 25)]
0 referencias | 0 cambios | 0 autores, 0 cambios
public void SumaTest(int num1 , int num2, int resultado)
{
    _cal = new Calculo();
    Assert.Equal(_cal.Suma(num1,num2),resultado);
}
(parámetro) int num1
1 referencia | 0 cambios | 0 autores, 0 cambios
```

En este caso si debemos definir los parámetros del método no como con el Test2 de .Net

Otra opción para pasar varios parámetros es utilizar una lista de objetos pero deberemos cambiar la etiqueta de atributo por [MenberData(nameof(List))]

```
1 referencia | 0 cambios | 0 autores, 0 cambios
public static IEnumerable<object[]> Data =>
new List<object[]>
{
    new object[] { 8, 5, 3 },
    new object[] { 10, 6, 4 },
    new object[] { 15, 7, 8 },
};
[Theory]
[MemberData(nameof(Data))]
0 referencias | 0 cambios | 0 autores, 0 cambios
public void RestaTest(int num1, int num2, int resultado)
{
    _cal = new Calculo();
    Assert.Equal(_cal.Resta(num1, num2), resultado);
}
```

Como vemos nos realiza los test de los métodos comprobando todas las causalisticas: