

5 Nitelikler (Attributes) ve Reflection (Yansıma)

Nitelikler sınıf, metot ya da diğer üye elemanlara ekstra bilgiler eklemek için bu elemanların önüne köşeli parantezler arasında yazılan bildirimlerdir.

Üretilen **assembly** içerisinde yer alan tip ve üyelere ekstra bilgiler katabilmeyi sağlarlar. Bir başka deyişle, **metadata** içerisine ilave bilgiler ekler.

System.Reflection (Yansımalar) sınıfındaki metotlar kullanılarak bu nitelikler çalışma zamanında elde edilebilir.

Örnek tanımlama:

Aşağıda metot isimli metoda nitelik ekleme gösterilmektedir.

```
[Nitelik (Argümanlar)]  
Public void metot() {  
}
```

2 tür nitelik vardır:

- Önceden tanımlanmış nitelikler (pre-defined),
- Kullanıcı tanımlı nitelikler (custom built).

5.9 Önceden Tanımlanmış Nitelikler

C# dilinde önceden tanımlanmış (pre-defined) bazı nitelikler vardır.

5.9.1 Conditional Attribute

Sadece metotlara uygulanan ConditionalAttribute; program kodu içindeki belirli bir metodu derleme esnasında dâhil edip etmeyeceğimizi bildirmek için kullanılır.

Bunun için **define** parametresinden faydalanırız.

Bu nitelik System.Diagnostics isim uzayında bulunur.

Örnek:

```
//#define ATLA //Bu satırı açarsak 3 metot ta çalışır.
```

```

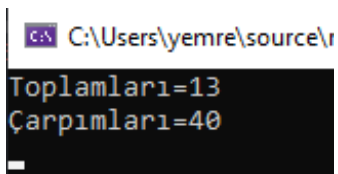
using System;
using System.Diagnostics;

namespace Attributes
{
    class C_Islemler
    {
        public void topla(int a, int b)
        {
            Console.WriteLine("Toplamları=" + (a+b));
        }
        [Conditional ("ATLA")]
        public void cikar(int a, int b)
        {
            Console.WriteLine("Farkları=" + (a - b));
        }
        public void carp(int a, int b)
        {
            Console.WriteLine("Çarpımları=" + (a * b));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            C_Islemler islem = new C_Islemler();
            int x = 5, y = 8;
            islem.topla(x, y);
            islem.cikar(x, y);
            islem.carp(x, y);
            Console.ReadKey();
        }
    }
}

```

Çıktısı:



```

C:\Users\yemre\source\l
Toplamları=13
Çarpımları=40

```

Sadece DEBUG modunda çalışmasını istediğimiz kodları bu yöntemle etiketleyebiliriz. Yukarıdaki kodu aşağıdaki gibi yazarsak program DEBUG modunda çalışırken `cikar` metodu çalışır ama RELEASE modunda çalışırken bu method devre dışı kalır.

Örnek:

```
using System;
using System.Diagnostics;

namespace Attributes
{
    class C_Islemler
    {
        public void topla(int a, int b)
        {
            Console.WriteLine("Toplamları=" + (a+b));
        }
        [Conditional ("DEBUG")]
        public void cikar(int a, int b)
        {
            Console.WriteLine("Farkları=" + (a - b));
        }
        public void carp(int a, int b)
        {
            Console.WriteLine("Çarpımları=" + (a * b));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            C_Islemler islem = new C_Islemler();
            int x = 5, y = 8;
            islem.topla(x, y);
            islem.cikar(x, y);
            islem.carp(x, y);
            Console.ReadKey();
        }
    }
}
```

5.9.2 Obsolete Attribute

Obsolete: eski, kullanılmayan. Bir metodun çağırılması durumunda uyarı veya hata mesajı verilmesini sağlar.

System isim uzayında bulunur.

Örnek:

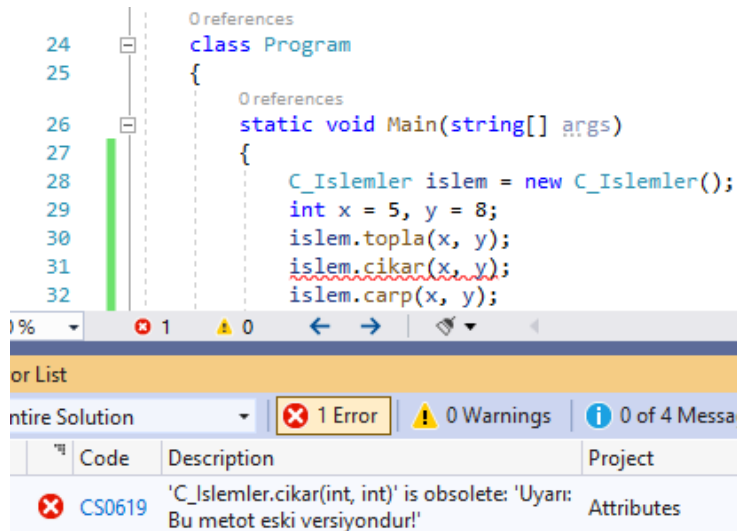
```
using System;
namespace Attributes
{
    class C_Islemler
    {
        public void topla(int a, int b)
        {
            Console.WriteLine("Toplamları=" + (a+b));
        }
        [Obsolete ("Uyarı: Bu metot eski versiyondur!", true)]
        public void cikar(int a, int b)
        {
            Console.WriteLine("Farkları=" + (a - b));
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            C_Islemler islem = new C_Islemler();
            int x = 5, y = 8;
            islem.topla(x, y);
            islem.cikar(x, y);
            Console.ReadKey();
        }
    }
}
```

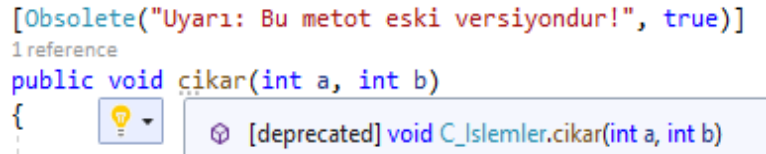
Niteliği true kullanmadan yazarsak program çalışır ve Warning üretir:

```
[Obsolete ("Uyarı: Bu metot eski versiyondur!")].
```

Hata mesajı yapılırsa çalışmayı engeller:

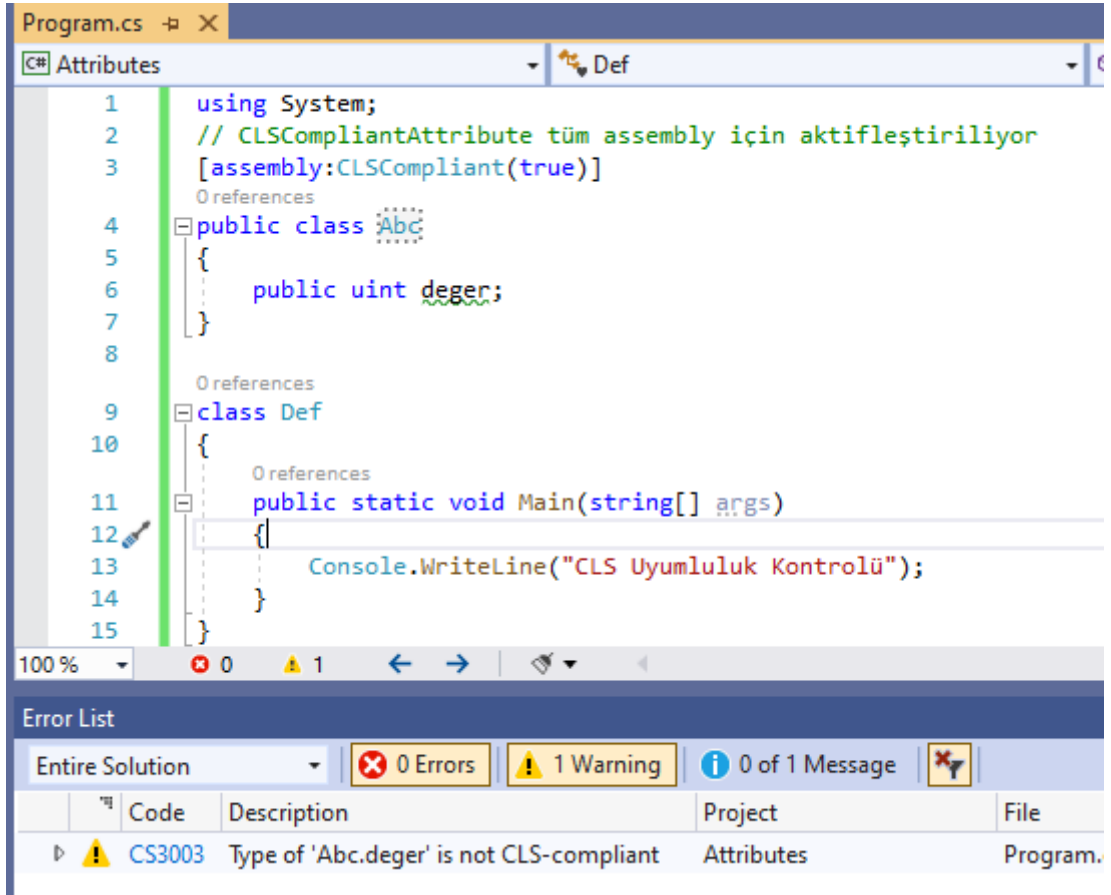


Obsolete attribute uygulanan metotlar artık deprecated (modası geçmiş) olarak tanınır:



5.9.3 CLSCompliantAttribute Attribute

Bu nitelik, belirli bir kod ögesinin Ortak Dil Belirtimi ile uyumlu olup olmadığını gösterir. Kod ögesi Ortak Dil Belirtimi ile uyumlu değilse, derleyici tarafından bir uyarı mesajı verilir.



6 Reflection (Yansıma)

Reflection, bir koddaki türlerin (types), yöntemlerin (methods) ve alanların (fields) meta verilerini tanımlama sürecidir. `System.Reflection` isim uzayı; yüklenen derlemeler (assembly), bunların içindeki sınıflar, yöntemler ve değer türleri gibi öğeler hakkında veri elde etmenizi sağlar. `System.Reflection`'ın yaygın olarak kullanılan sınıflarından bazıları şunlardır:

Sınıf	Tanımı
Assembly	Ortak bir dil çalışma zamanı (common language runtime) uygulamasının yeniden kullanılabilir, sürümlenebilir ve kendi kendini tanımlayan yapı taşı olan bir derlemeyi (assembly) tanımlar.
AssemblyName	Bir derlemeyi benzersiz bir adla tanımlar.
ConstructorInfo	Bir sınıf yapıcısını tanımlar ve meta verilere erişim sağlar.
MethodInfo	Sınıf yöntemini tanımlar ve meta verilerine erişim sağlar.
ParameterInfo	Bir yöntemin parametrelerini tanımlar ve meta verilerine erişim sağlar.
EventInfo	Olay bilgilerini tanımlar ve meta verilerine erişim sağlar.
PropertyInfo	Bir property'nin niteliklerini tanımlar ve meta verilerine erişim sağlar.
MemberInfo	Bir üyenin nitelikleri hakkında bilgi alır ve üye meta verilerine erişim sağlar.

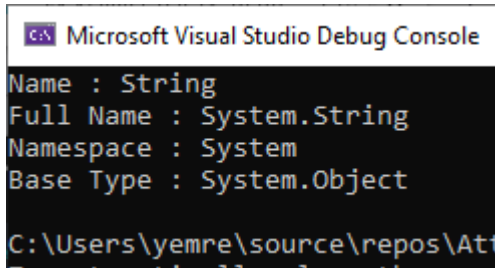
Örnek:

Aşağıda verilen kodda `typeof` metodunu kullanarak `type t`'yi `string` olarak yüklüyoruz. Ardından `string` sınıfı hakkında; adı (`name`), tam adı (`fullname`), isim uzayı (`namespace`) ve temel türü (`basetype`) gibi bilgileri bulmak için `t` üzerinde reflection uygulanıyor:

```
using System;
namespace Reflection
{
    class Program
    {
        static void Main(string[] args)
        {
            // t yi string tipinde başlat
            Type t = typeof(string);

            // t ile ilişkili bilgilere yansıma ile ulaş
            Console.WriteLine("Name : {0}", t.Name);
            Console.WriteLine("Full Name : {0}", t.FullName);
            Console.WriteLine("Namespace : {0}", t.Namespace);
            Console.WriteLine("Base Type : {0}", t.BaseType);
        }
    }
}
```

Çıktısı:



```
Microsoft Visual Studio Debug Console
Name : String
Full Name : System.String
Namespace : System
Base Type : System.Object
C:\Users\yemre\source\repos\Att
```

Örnek:

Aşağıdaki kodda programa ilişkin sınıflar, metotlar ve parametreleri içeren tüm metadata bilgilerini reflection ile gösteriyoruz:

```
using System;
using System.Reflection;

namespace Reflection_Metadata
{
    // Student sınıfı tanımla
```

```

class Student
{
    // Property lerin tanımlanması
    public int RollNo
    {
        get;
        set;
    }

    public string Name
    {
        get;
        set;
    }

    // Parametresiz Constructor
    public Student()
    {
        RollNo = 0;
        Name = string.Empty;
    }

    // Parametrelili Constructor
    public Student(int rno, string n)
    {
        RollNo = rno;
        Name = n;
    }

    // Method to Display Student Data
    public void displayData()
    {
        Console.WriteLine("Roll Number : {0}", RollNo);
        Console.WriteLine("Name : {0}", Name);
    }
}

class GFG
{
    static void Main(string[] args)
    {
        // Assembly sınıfının örneğini tanımla
        // aktif assembly yi yüklemek için GetExecutingAssembly
        metodu çağır
        Assembly executing = Assembly.GetExecutingAssembly();

        // assembly nin tiplerini tutmak için dizi tanımla
        Type[] types = executing.GetTypes();
    }
}

```



```

foreach (var item in types)
{
    // Her bir tipi göster
    Console.WriteLine("Class : {0}", item.Name);

    // metotları tutmak için dizi tanımla
    MethodInfo[] methods = item.GetMethods();
    foreach (var method in methods)
    {
        // Her bir metodu göster
        Console.WriteLine("--> Method : {0}",
method.Name);

        // parametreleri tutmak için dizi tanımla
        ParameterInfo[] parameters =
method.GetParameters();
        foreach (var arg in parameters)
        {
            // Her bir parametreyi göster
            Console.WriteLine("----> Parameter : {0}
Type : {1}",
                arg.Name, arg.ParameterType);
        }
    }
}
}
}
}

```

Çıktısı:

```
Seç Microsoft Visual Studio Debug Console

Class : Student
--> Method : get_RollNo
--> Method : set_RollNo
----> Parameter : value Type : System.Int32
--> Method : get_Name
--> Method : set_Name
----> Parameter : value Type : System.String
--> Method : displayData
--> Method : GetType
--> Method : ToString
--> Method : Equals
----> Parameter : obj Type : System.Object
--> Method : GetHashCode
Class : GFG
--> Method : GetType
--> Method : ToString
--> Method : Equals
----> Parameter : obj Type : System.Object
--> Method : GetHashCode

C:\Users\yemre\source\repos\Reflection\bin\Debu
```