Attributes (Nitelikler)

- Nitelikler, uygulandıkları classes, methods, structures, enumerators veya assemblies gibi programınızın parçalarının çalışma zamanındaki davranışlarının değiştirilmesine olanak sağlayan sınıflardır.
- Attribute, abstract bir sınıftır. Dolayısıyla örneklenemez ancak bir nitelik sınıfının içermesi gereken temel üyeleri bünyesinde barındırır.
- .Net içinde mevcut olan veya bizim tarafımızdan geliştirilen nitelikler daima Attribute sınıfından türemek zorundadırlar.
- Üretilen assembly içerisinde yer alan tip ve üyelere ekstra bilgiler katabilmeleridir. Bir başka deyişle, metadata içerisine ilave bilgiler eklenebilmesini sağlamaktadır.
- Bu metadata bilgileri üretilen assembly' lar içerisinde yer alır ve yansıma(Reflection) teknikleri ile çalışma zamanında değerlendirilebilir.
- Bu konu ile ileri de çok karşılaşacaksınız. Hızlı kod üretmede kullanabilirsiniz.

Attributes (Nitelikler)

- .Net Framework'te Niteliklerin iki tip sağlar:
  - pre-defined attributes (Önceden tanımlanmış nitelikler)
    - AttributeUsage
    - Conditional
    - Obsolete
  - custom built attributes (Özel yapılmış nitelikler)
- Nitelikler «[…]» karakterleri arasında çağrılır. Kullanımı:

  [attribute(positional_parameters, name_parameter = value, ...)]

  element

- element: bir class, method, structure, enumerator veya assembly olabilir. Positional_parameters gerekli bilgileri belirtir ve name_parameter istediği bilgileri belirtir.

AttributeUsage

- AttributeUsage kişisel attribute class nasıl olacağını tanımlar. Item'ların type'larına hangi attribute'un uygulanabileceğini belirler.

```
[AttributeUsage(
   validon,
   AllowMultiple=allowmultiple,
   Inherited=inherited
)]
```

- validon parametresi dil elementlerinin hangi attritube yerleştirilebileceğini belirler. Bir enumerator AttributeTargets değerinin bir kombinasyonudur. Geçerli değeri AttributeTargets.All'dur.
- allowmultiple parametresi (Seçimlik - optional) bir Boolean değerdir, bu attritube'un AllowMultiple property değer sağlar. Eğer true ise , attritube çok amaçlıdır. Geçerli değeri false'tur (tek amaçlı).
- inherited paramtresi (Seçimlik) bir Boolean değerdir, bu attritube'un Inherited property değer sağlar. Eğer true ise, derived sınıflar tarafından kalıtım uygulanabilir. Geçerli değeri false'tur (Kalıtım uygulanamaz.).

AttributeUsage

- Kullanımı

```
[AttributeUsage(
   validon,
   AllowMultiple=allowmultiple,
   Inherited=inherited
)]
```

- Örneğin:

```
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Feild |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]
```

AttributeUsage

```
// Bir attritube sınıflara eklenebilir
[AttributeUsage(AttributeTargets.Class)]
public class PerlsAttribute : Attribute
{
}

//Üstteki sınıfın kısa sözdizimi ile çağrımı
[Perls]
class Example1
{
}

//Üstteki sınıfın uzun sözdizimi ile çağrımı
[PerlsAttribute]
class Example2
{
}

class Program
{
    static void Main()
    {
        //Derleme yapılabilmesi için
    }
}
```

Attritube sınıfı olduğu belirleniyor.

Attritube sınıfından kalıtım uygulanıyor

PerlsAttribute sınıfın attribute özellikleri Example1 ve Example2 sınıflarına aktarılmıştır.

PerlsAttribute yerine Attribute kelimesi kullanılmadan sadece Perls yazılabilir.

AttributeUsage - Constructor

```
// Attribute
[AttributeUsage(AttributeTargets.Class)]
public class PerlsAttribute : Attribute
{
    //Veri depolamak için bir değişken
    string _id;
    //Attritube'in Constructor'ı
    public PerlsAttribute(string id)
    {
        this._id = id;
    }
    //Id getir
    public string Id
    {
        get { return this._id; }
    }
}

//Attritube'i uygula
[Perls("Dot")]
class Example2
{
}
```

constructor

Id değişkenine constructor üzerinden Attribute değeri gönderme

Attribute - Property

```
// Attribute
[AttributeUsage(AttributeTargets.Class)]
public class Perls3Attribute : Attribute
{
    //Veri depolamak için bir değişken
    string _id;
    //Attritube'in Constructor'ı boş
    public Perls3Attribute()
    {
    }
    //Id getir ve değiştirebilir (get / set)
    public string Id
    {
        get { return this._id; }
        set { this._id = value; }
    }
}

//Attritube'i uygula
[Perls3Attribute(Id = "Sam")]
class Example3
{
}
```

Constructor boş

Id bunun üzerinden ulaşırız.

Id değişkenine property üzerinden Attribute değeri gönderme

Conditional

- Conditional (Şartlı) method her zaman çalışmaz.
- #define ile tanımlanan belirli bir ifade sadece derlenir.
- #undef ile istenilmeyen kısımlar çalışmaz.
- #define ve undef işlemi kodun üst kısmına yapılır.
- Conditional söz dizimi:

```
[Conditional(
    conditionalSymbol
)]
```

Conditional

```
#define PERLS
#undef DOT
using System;
using System.Diagnostics;

namespace Ders09_07_Conditional
{
    class Program
    {
        static void Main()
        {
            MethodA();
            MethodB();
            Console.ReadKey();
        }

        [Conditional("PERLS")]
        static void MethodA()
        {
            Console.WriteLine("MethodA çalıştı.");
        }

        [Conditional("DOT")]
        static void MethodB()
        {
            Console.WriteLine("MethodB çalıştı.");
        }
    }
}
```

Tanımlı kısım. Derlenir.

Tanımlanmayan kısım. Derlenmez.

Çalışır

Çalışmaz

MetdodB'yi çalıştırmamak için normal şartlarda main'den silmelisiniz. Fakat yazılım karmaşıklaştıkça bu silme süreci uzayabilir. Bazen derlemek istemediğiniz kısım halen geliştirilmekte olabilir. Bu durumlarda Conditional Metot çok işe yarar.

Obsolete (eski)

- Bu predefined attritube kullanılmayacak olan program varlığını işaretler.
- Belirli bir elementi compiler tarafından derlenmeyeceğine dair bilgi vermeye olanak tanır.
- Örneğin, yeni bir metot yazdığınızda hala eski metodunuzu görmek istiyorsanız (silmek istemiyorsanız), fakat takım arkadaşlarınızı (veya ileride kendinizi) uyarmak için kullanılabilir.

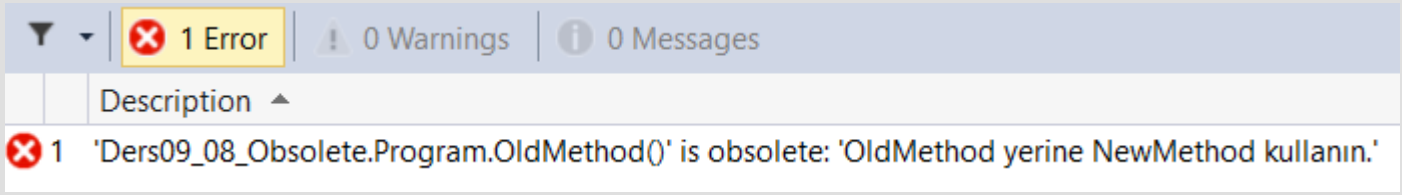
```
[Obsolete(
   message
)]
[Obsolete(
   message,
   iserror
)]
```

- *message* parametresi niye obsolote işleminin yapıldığının tutulduğu string türünden kısımdır.
- *İserror parametresi* Boolean bir değerdir. (seçimliktir) Eğer doğru ise derleyici kodu derlemez. Eğer yanlış ise sadece uyarı (warning) oluşturur.

Obsolete

```
class Program
{
    [Obsolete("OldMethod yerine NewMethod kullanın.", true)]
    static void OldMethod()
    {
        Console.WriteLine("Eski Metot");
    }
    static void NewMethod()
    {
        Console.WriteLine("Yeni Metot");
    }
    public static void Main()
    {
        OldMethod();
        Console.ReadKey();
    }
}
```

Parametre true olduğu için çalışmadı. Aşağıdaki uyarıyı verdi.



Parametre false olduğunda sadece uyarı verir. Warning message'larının derleyicinin çalışmasını etkilemediğini söylemiştik. (Örneğin değişken tanımlayıp hiç kullanmama durumda warning oluşur.)

Custom Built Attributes

- .Net Frawrok'ü bildirimli bilgi depolama ve çalışma zamanında bunları almak için kendinize özel Attribute'lerde tanımlamanıza olanak sağlar.
- Bu bilgi kod tasarımıza ve uygulama ihtiyaçlarına bağlı olarak herhangi bir hedef elemente ilişkili olabilir.
- Kişisel Attribute yaratma ve kullanma 4 adımda içerir.
  - Bir custom attribute tanımlama
  - Custom attribute'i inşa etme
  - Hedef program elementi'ne Custom attribute uygulama
  - Reflection üzerinden Attribute'lere erişim
- Son adım, çeşitli gösterimler bulmak için metadata üzerinden okumak için basit bir program yazmayı içerir. Metadata diğer verileri tanımlamak için kullanılan veri veya bilgi ile ilgili verilerdir. Bu program zamanında özelliklerini erişmek için reflections (yansımaları) kullanmalısınız.
- Reflection konusu önümüzdeki haftalarda anlatılacaktır.

Custom Built Attributes

```
//custom attribute'ın bir sınıf ve üyelerine atanacak
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]
```

Oluşturulacak Attribute Class'ın erişimleri belirleniyor.

Attribute Class'ından kalıtım yapılıyor.

İlk iki aşama

```
public class DeBugInfo : System.Attribute
{
    private int bugNo;
    private string developer;
    private string lastReview;
    public string message;
    public DeBugInfo(int bg, string dev, string d)
    {
        this.bugNo = bg;
        this.developer = dev;
        this.lastReview = d;
    }
    public int BugNo
    {
        get{ return bugNo; }
    }
    public string Developer
    {
        get{ return developer; }
    }
    public string LastReview
    {
        get{ return lastReview; }
    }
    public string Message
    {
        get { return message; }
        set { message = value;}
    }
}
```

Custom Built Attributes

```
[DeBugInfo(45, "Zara Ali", "12/8/2012", Message = "Return type mismatch")]
[DeBugInfo(49, "Nuha Ali", "10/10/2012", Message = "Unused variable")]
class Rectangle
{
    //Değişkenler
    protected double length;
    protected double width;
    public Rectangle(double l, double w)
    {
        length = l;
        width = w;
    }
    [DeBugInfo(55, "Zara Ali", "19/10/2012", Message = "Return type mismatch")]

    public double GetArea()
    {
        return length * width;
    }
    [DeBugInfo(56, "Zara Ali", "19/10/2012")]

    public void Display()
    {
        Console.WriteLine("Length: {0}", length);
        Console.WriteLine("Width: {0}", width);
        Console.WriteLine("Area: {0}", GetArea());
    }
}
```

3. Adım

Constructor 3 parametreli olmasına rağmen 4 parametreli tanımlamalar vardır. Message property'si set olduğu için erişilebilmiştir. Diğer değişkenlere constructor üzerinden erişilmiştir.