

The screenshot shows a Visual Studio interface with three code files in a solution:

- File 1 (Top):** A file named "Program.cs" containing the following C# code:

```
using Project;
internal class Program
{
    private static void Main()
    {
        Console.WriteLine(Taha.Text);
        Console.WriteLine(Taha.Text);
    }
}
```
- File 2 (Middle):** A file named "Taha.cs" containing the following C# code:

```
namespace Project
{
    class Taha
    {
        public const string Text = "Taha2";
    }
}
```
- File 3 (Bottom):** A file named "Taha.cs" containing the following C# code:

```
namespace Project
{
    class Taha
    {
        public const string Text = "Taha1";
    }
}
```

The code in File 1 and File 2 is identical. The code in File 3 is identical except for the value of the `Text` constant.

Each code file has its own status bar at the bottom indicating "No issues found". The status bar for File 3 also includes the handle "@csharp_gunlukleri".

```
namespace Project
{
    2 references
    class Taha
    {
        public const string
            Text = "Taha2";
    }
}
```

```
namespace Project
{
    2 references
    file class Taha
    {
        public const string
            Text = "Taha1";
    }
}
```

Burada class'ları hazırladıktan sonra Taha.Text dediğimizde artık "Taha2" değeri gelecek. File özelliği class'ı sadece o dosya içinde kullanacak şekilde ayarlar. Genelde isimlendirmelerin düzenli olması gerekiğinde ihtiyaç duyulur.

Ayrıca illa aynı isimle class olması gerekmez. sadece o dosya içinden erişilmesi istenirse file özelliği kullanılabilir. Bir nevi public, protected, private gibi bir özellik. Sadece o dosya içinden erişilmek istenirse file eklenir.

The screenshot shows two code editors side-by-side. Both editors have the same code:`0 references
private static void Main(string[] args)
{
 TahaPartial.
 ★ ReferenceEquals
 ★ Taha1
 ★ Taha2
 ★ Equals
 Equals
 ReferenceEquals
 Taha1
 Taha2
 [+] [] []
Project.T
espace Project

2 references
public partial class TahaPartial
{
 public const string Taha2 = "Taha2";
}
Project.T
espace Project

2 references
public partial class TahaPartial
{
 public const string Taha1 = "Taha1";
}
Project.T`The word 'partial' in the first editor's class definition is highlighted with a red rectangle. In the second editor, the entire class definition is highlighted with a red rectangle.

Partial ise file etiketinin tersi olarak düşünülebiliriz. Tam öyle değil ama anlamak için. Partial etiketi ile bir class'ı birden fazla dosyaya bölebilirisiniz.

Ben genelde sabit değerleri tek class içinde tutuyorum. Kategori etmek için region yerine partial class içinde dosya dosya bölüyorum. Yani BackendConstant.cs, FrontendConstant.cs dosyalarına partial olarak tek class koyuyorum. Constant adında.

Büyük classlarda kontrol sağlamak için kullanabilirisiniz.

```
0 references
private static void Main(string[] args)
{
    Veril veri = new Veril("Taha", 27);

    veri.isim = "Ahmet";

    veri = veri with { isim = "Ahmet" };
}
```

```
namespace Project

2 references
public record Veril(string isim, int yas);

0 references
public record Veri2
{
    0 references
    public string isim { get; init; } = string.Empty;
    0 references
    public int yas { get; init; }
}
```

Record class'lar temelde bir kere oluşturulan ve değiştirilemeyen classlardır. İki farklı üretme yolu vardır. `veri.isim = "Ahmet";` satırında bverilen hata gibi. Sonradan veri değiştirmek yasak.

Kolaylık olması açısından syntax içinde with operatörü kullanılır. Bir record değişkeni yanına with yazarak, sonrasında süslü parantez açarak değiştirmek istediklerimizi yazar ve böyle değiştiririz.

Şimdi ne fark var derseniz. Doğrudan değişken içindeki değer değiştirilemez. With ile aslında arkapanda tekrar bir record class üretilir ve o sıra yeni değerler ile üretilir.

Ben veritabanına gönderilecek veya veritabanından alınan verileri record ile tutarak olası değişimleri geliştiriciler arasında belirtiyorum.