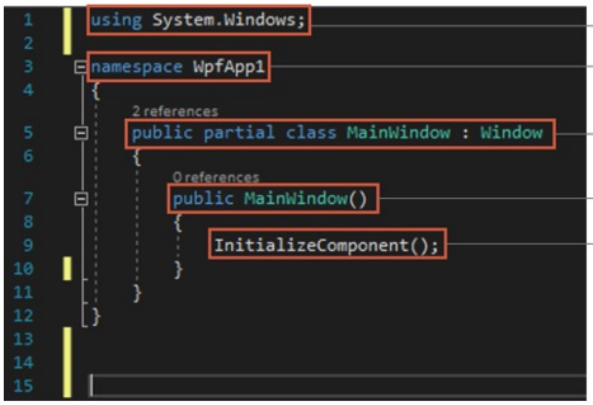

GÖRSEL PROGRAMLAMA
Temel C# Sözdizimi

Uygulama Girişi



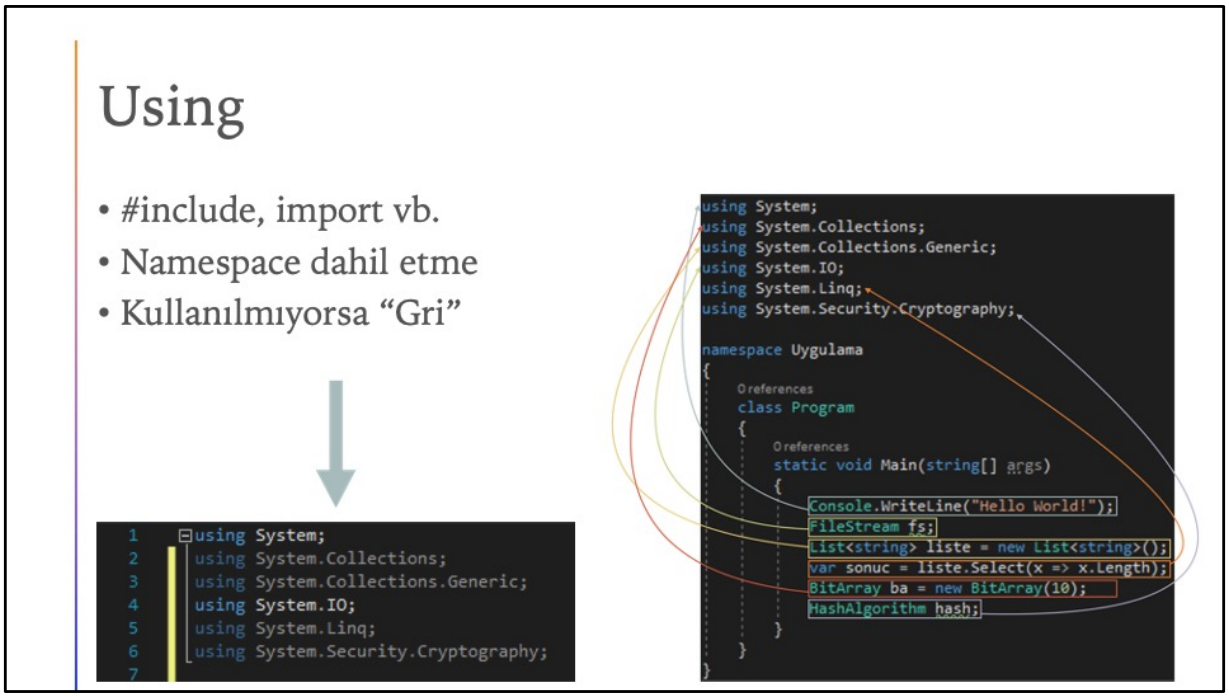
Kullanılacak kütüphanelerin eklenmesi

İsimuzayı

Sınıf

Başlangıç metodu

Tüm bileşenlerin oluşturulması ve başlatılması



Kütüphanelerin kullanabilmesi için isimlerinin using kelimesi ile koda eklenmesi gerekmektedir. C dilindeki #include ve python dilindeki import örneklerine benzetilebilir.
Visual studio’da eğer bir kütüphane kullanılmadığı halde using ile projeye dahil edildiyse bu satırlar gri renkte gözükecektir.

Using

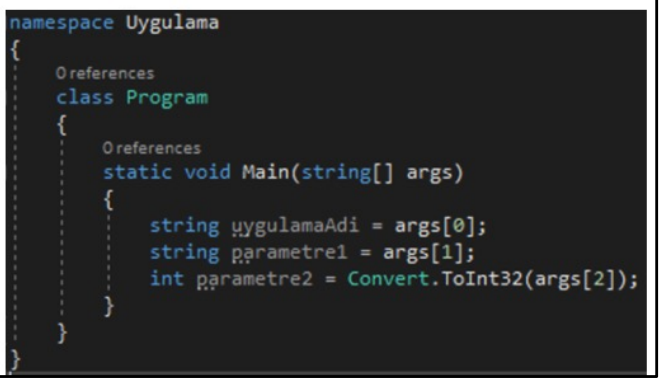
- Kod içerisinde
- Dispose

```
namespace Uygulama
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            using(FileStream fs = new FileStream("DosyaYolu", FileMode.Open))
            {
                fs.WriteByte(1);
            }
            fs.WriteByte(1);
        }
    }
}
```

using anahtar kelimesi kod içerisinde de kullanılabilmektedir. Using bloğu içerisinde tanımlanan değişkenler kullanılır ve blok bitiminde değişkenler dispose edilir. Bu değişkenlere erişim mümkün değildir.
Bunun amacı değişkenlerin garbage collector'a verilmesi için metodun bitmesinin beklenmesine gerek olmadığını belirtmektir.

Main

- static
- void
- string[] args



```
namespace Uygulama
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            string uygulamaAdi = args[0];
            string parametre1 = args[1];
            int parametre2 = Convert.ToInt32(args[2]);
        }
    }
}
```

Konsol uygulamalarında main metodu c, java ve c++'ta olduğu gibi uygulamanın başlangıç noktasıdır. args parametresi ile dışarıdan parametre alınarak uygulama çalıştırılabilmektedir. Wpf uygulamasında ise main metodu build işleminde otomatik olarak arkaplanda oluşturulur.

Değişkenler

- bool
- byte (sbyte)
- short (ushort)
- int (uint)
- long (ulong)
- float
- double
- char
- decimal

temel değişkenler null değeri almazlar (C# 7'den itibaren tür? gibi nullable kullanılabilmektedirler).
bool türü yalnızca true ya da false değeri saklarlar. Buna rağmen bellekte 1 bit değil 1 byte yer kaplamaktadırlar.
Byte türü işaretsiz olarak kullanılır. 0-255 arası değerler atanabilmektedir. -128 +127 arası değerler kullanabilmek için sbyte kullanılabilir.
Short, int ve long türleri byte'ın aksine işaretlidir ve bu türlerin işaretsiz halleri için unsigned (u) kullanılmalıdır.
Char ve byte türleri C# dilinde C dilinden farklı olarak ayrı anlamlar taşımaktadır. Byte türü 1 byte'lık sayı değerlerini saklarken, char türü karakter değişkenler saklamaktadır ve sayısal işlemleri desteklememektedir.

Operatörler

Kategori	Operatörler
Aritmetik	-, +, *, /, %, ++, --
Mantıksal	&&, ||, !, ^
İkili	&, |, ^, ~, <<, >>
Karşılaştırma	==, !=, >, <, >=, <=
Atama	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
String birleştirme	+
Tip Dönüşümü	(□□□), □□, □□, □□□□□□, □□□□□□
Diğer	., □□□, (), [], ?:, ??

C# dilinde her dilde olduğu gibi aritmetik, mantıksal, binary ve karşılaştırma operatörleri bulunmaktadır.
+ operatörü stringler için de desteklenmektedir.
Bir türden diğerine dönüşüm için değişken başına (tür) eklenebilir. (int x = (int)12.1203 gibi)
As anahtar kelimesi ile bir değişkeni belirli bir türden kullanmak mümkündür (int a = x as int gibi)
Typeof bir değişkenin türünü, sizeof ise boyutunu döndürmektedir.
?? Null olabilen değişkenler için null kontrolü gerçekleştirir (a ?? 1 ifadesi a null değilse a, null ise 1 değerini döndürür.)

Koşullar

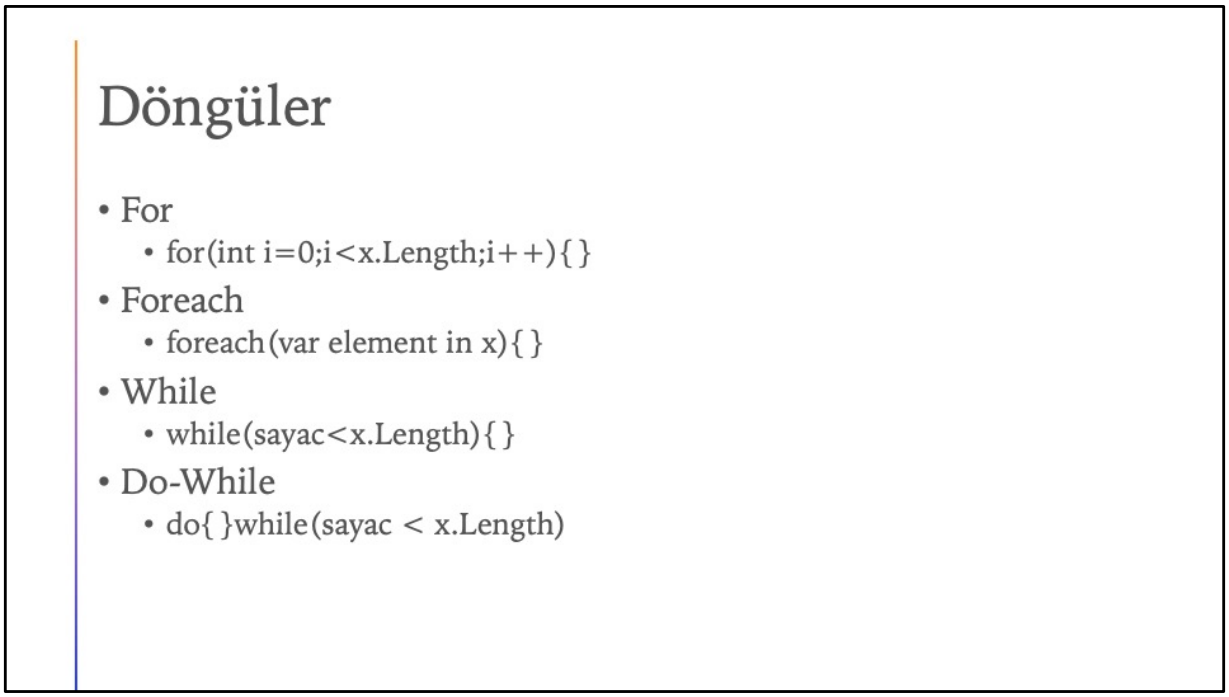
- if
  - else if
  - else
- switch
  - case
  - default

If anahtar kelimesi parantez içerisinde verilen koşulun sağlanması durumunda blok içerisindeki işlemleri gerçekleştirir. Diğer dillerde olduğu gibi koşulun sağlanmadığı durumda yönlenilen else if ve else blokları da bulunmaktadır.
Switch ifadesi ile belirli verilen bir parametrenin değerine göre belirli koşullara dallanma işlemi gereçekleştirilebilmektedir. Genellikle bir seçim durumunda değişkenin değerinin kontrolünde kullanımı daha çok yaygındır.

Diziler

- Tanım
  - int[] x;
  - int[] x = new int[3];
  - int[] x = new int[3]{1,2,3};
    - int[] x = new int[]{1,2,3};
    - int[] x = {1,2,3};
  - var x = new int[3]{1,2,3};
    - var x ={1,2,3};
- Erişim
  - x[0] = 3;
  - int a = x[1];
- Boyut
  - x.Length

Diziler birden farklı şekilde tanımlanabilirler. İlk değer ataması { } parantezleri içerisinde yapılabilir. Bu durumda boyut verilmesine gerek yoktur. Tür verilmediğinde de çözümlemeyi yine derleyici yapacaktır.



C# üzerinde 4 farklı döngü kullanılabilmektedir. Belirli bir sayı değeri aralığında tekrarlı gerçekleştirilecek işlemler için for döngüsü kullanılmaktadır. Belirli bir listenin elemanları üzerinde gezmek veya her elemana sırasıyla erişmek için foreach ifadesi kullanılmaktadır. Bir koşulun sağlanmasına kadar olan adımlarda tekrarlı bir şekilde gerçekleştirilen işlemler için while döngüsü kullanılır. Do while'ın farkı ise bu işlemleri ilk seferde gerçekleştirip kontrolü sona bırakmasıdır. Teknik olarak bir döngü diğerlerinin yerine kullanılabilmektedir.

Yapılar

- struct kelimesi
- struct isim{erişim belirleyicisi değişken türü değişken adı}

```
struct x
{
    public string alan1;
    public int alan2;
    public double alan3;
}
```

- Örneği alınabilir / değer türünde kullanılabilir

```
static void Main(string[] args)
{
    x yapi1;
    x yapi2 = new x();
    yapi1.alan1 = "";
    yapi2.alan1 = "";
}
```

- Yapıcı ve metot içerebilir

```
struct x
{
    public string alan1;
    public int alan2;
    public double alan3;
    0 references
    public x(string alan1,int alan2,double alan3)
    {
        this.alan1 = alan1;
        this.alan2 = alan2;
        this.alan3 = alan3;
    }
    0 references
    public string dondur()
    {
        return alan1;
    }
}
```

C dilinde olduğu gibi C# dili de yapıları desteklemektedir. Yapıların sınıflardan farkı değer türünde olmasıdır. Yapılar sınıflarda olduğu gibi yapıcı veya metotlar içerebilmektedir.

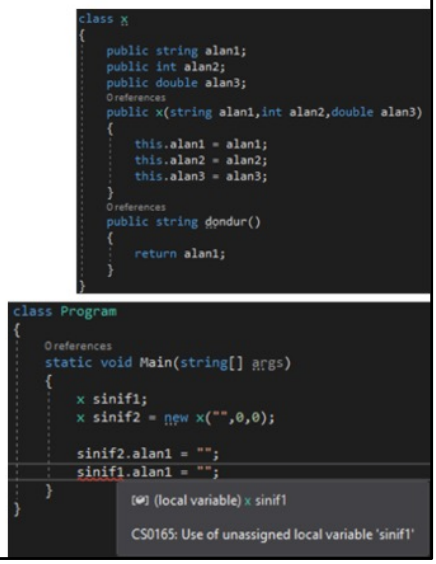
New

- Türün örneği
- Heap
- Çöp toplayıcı
- Temel türler / değer türünde struct

Temel türler ve değer türünde struct'ta kullanılmaz.
New ile alınan değişkenler heap'te oluşturulur.
Referansların kullanılmadığının tespiti durumunda garbage collector çalışır.

Sınıflar

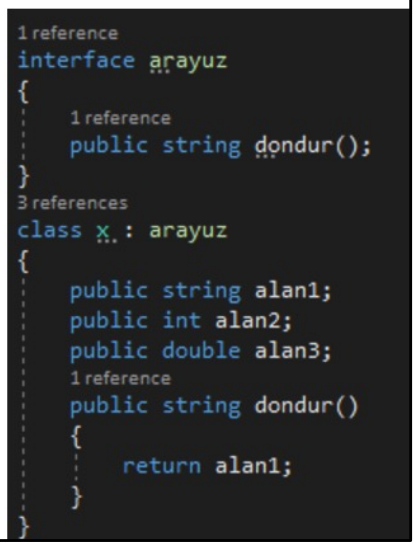
- class kelimesi
- Her zaman heap'te
- Örnek alınmalı (new)



C#'ta sınıfların tanımlanması için class anahtar kelimesi kullanılır. Yapılardan farklı olarak referans türündedirler ve new ile alınırlar. Nesneye yönelik programlamada çoklu kalıtım haricindeki tüm paradigmalar C#'ta desteklenmektedirler.

Arayüzler

- Şablonlar
- Sınıflar tarafından içerilir
- İçeren sınıfların implementasyonu



```
1 reference
interface arayuz
{
    1 reference
    public string dondur();
}
3 references
class x : arayuz
{
    public string alan1;
    public int alan2;
    public double alan3;
    1 reference
    public string dondur()
    {
        return alan1;
    }
}
```

Arayüzler, arayüzü implement edecek sınıflar için bir kontrat oluştururlar. İmplementasyon içermezler. Yalnızca imza içerirler.

Try-Catch-Finally

- Hata yakalama
- try - çalıştırma
- catch - hata
- finally - her durum sonunda

```
static void Main(string[] args)
{
    int y = 0;
    int x = 0;
    try
    {
        x = 0 / y; //Sıfıra bölme
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);//Hata durumunda
    }
    finally
    {
        x = 20; //Her zaman
    }
}
```

Çalışma zamanında alınan hatalarda programın durmasını engellemek ve hataların kaydını tutabilmek amacıyla try catch kullanılabilmektedir. Try bloğunun içerisindeki kod çalıştırılırken oluşabilecek bir hatada catch bloğuna düşülür ve bu bloktaki kod çalıştırılır. Finally kısmı ise hata yakalanıp yakalanmamasından bağımsız olarak her zaman çalıştırılmaktadır.