

C# 8-11 újdonságok, Top level statements, Nullable



Répasné Babucs Hajnalka

Répás Csaba



.NET és C# verziószámok

.NET verzió	C# verzió
.NET 7.x	C# 11
.NET 6.x	C# 10
.NET 5.x	C# 9.0
.NET Core 3.x	C# 8.0
.NET Core 2.x	C# 7.0
.NET Standard 2.1	C# 8.0
.NET Standard 2.0	C# 7.3
.NET Standard 1.x	C# 7.3
.NET Framework	C# 7.3



.NET 6 és .NET 7 újdonságok



- A .NET 6 egyesíti az SDK-t, az alapkódtárakat és a futtatókörnyezetet mobil, asztali, IoT- és felhőalkalmazásokban.
 - Egyszerűsített fejlesztés, kisebb kód mennyiség
 - Jobb teljesítmény, webes alkalmazásoknál, felhőben is
 - Új Git-eszközök, intelligens kódszerkesztés
 - Robusztus diagnosztikai és tesztelési eszközök
- A .NET 7 a teljesítmény növelésére fókuszál
 - Reguláris kifejezések használatának bővítése
 - Bővített API szolgáltatások
 - .NET MAUI (Mutiplatform App UI): platformfüggetlen keretrendszer natív mobil- és asztali alkalmazások C# és XAML használatával történő létrehozásához





C# 8, 9, 10 és 11 újdonságai



➤ Readonly tagok kiterjesztése (struct, property, metódus)



➤ is, and, or, not

➤ Nullable érték típusok



➤ Indexelés újdonságai

➤ Record típus



➤ Interfész alapértelmezett implementációja



➤ Példányosítás újdonságai (egyszerűsített new())



➤ Global usings


➤ Lambda-kifejezés fejlesztése

➤ Stb.

Top level statements

- C# 9-től elérhető, VS 2022-ben alapértelmezett Console sablon

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```



```
using System;

namespace Application
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

- Csak a Main metódus törzsét kell megírni
- Implicit using: a fordító automatikusan hozzáadja a megszokott névtereket

Additional information

Console App

C#

Linux

macOS

Windows

Console

Framework ⓘ

.NET 7.0 (Standard Term Support)

☐ Do not use top-level statements ⓘ



Top level statements használata



- Minden projekt csak 1 top level statements fájlal rendelkezhet
 - Minden alkalmazás csak egy belépési ponttal rendelkezhet
 - Ne írd Main() nevű metódust!
- A szükséges (nem alapértelmezett) using-okkal kezdődjön a fájl
- A névttereket és a típusdefiníciókat a fájl végén lehet létrehozni
- Lehet hivatkozni az **args** tömbbel a beírt parancssori argumentumokra
- Lehet **return** kulcsszó után visszaadni értéket





Nullable reference types



- A referencia típusok eddig is felvehettek null értéket
 - Ha nem kezeltük le, és úgy próbáltunk rajta metódust hívni, vagy tulajdonságot használni, akkor **NullReferenceException** típusú kivételt kaptunk.
 - Leggyakoribb futásidejű hibajelenség
- Ha egy típusnál meg szeretnénk engedni, hogy null értéket vegyen fel, akkor ki kell tenni a típus után a **?** operátort
 - Ha elmulasztjuk a null ellenőrzést ?-es típusnál, akkor fordítási idejű figyelmeztetést kapunk
 - Ha nem jelezzük, hogy a változó null értéket is felvehet, és mégis null értéket kap, arra is fordítási idejű hibát kapunk





Nullable value types - deklaráció



- Nullable<T>-ből származnak
- Null értéket felvehető értéktípust a típus utáni ? használatával adhatunk meg



- Példák:

```
double? pi = 3.14;
```

```
char? betű = 'a';
```

```
int m2 = 10;
```

```
int? m = m2;
```

```
bool? flag = null;
```

```
// Tömb, ami nullable érték elemtípusú:
```

```
int?[] tomb = new int?[10];
```





Null értékű példány vizsgálata



- Null vizsgálathoz az **is** operátor is használható



```
int? a = 42;  
if (a is int valueOfA)  
{  
    Console.WriteLine($"a is {valueOfA}");  
}  
else  
{  
    Console.WriteLine("a does not have a value");  
}  
// Output:  
// a is 42
```





Nullable<T>.HasValue használata



- Ha a HasValue értéke false, akkor a Value-ra való hivatkozás **InvalidOperationException** kivételt dob



```
int? b = 10;
if (b.HasValue)
{
    Console.WriteLine($"b is {b.Value}");
}
else
{
    Console.WriteLine("b does not have a value");
}
// Output:
// b is 10
```





?? operátor – helyettesítő érték megadása



- Ha lehetséges null értékű értéket szeretnénk hozzárendelni egy nem null értékű típusú változóhoz, meg lehet adni, hogy null érték esetén helyette milyen értéket rendeljen hozzá



```
int? a = 28;  
int b = a ?? -1;  
Console.WriteLine($"b is {b}"); // output: b is 28
```

```
int? c = null;  
int d = c ?? -1;  
Console.WriteLine($"d is {d}"); // output: d is -1
```



- A mögöttes értéktípus alapértelmezett értéke is használható null helyett a `Nullable<T>.GetValueOrDefault()` metódus használatával



Aritmetikai operátorok használata nullable típusoknál

- Az aritmetikai operátoroknál ha az egyik operandus null, akkor az eredmény **null** lesz



```
int? a = 10;  
int? b = null;  
int? c = 10;
```

```
a++;           // a is 11  
a = a * c;     // a is 110  
a = a + b;     // a is null
```

Összehasonlító operátorok használata nullable típusoknál

➤ Összehasonlító operátoroknál, ha legalább az egyik operandus null, akkor az eredmény **false** lesz

➤ Az egyenlőség operátor esetén, ha mindkét operandus null, az eredmény **true** lesz

```
int? a = 10;
Console.WriteLine($"{a} >= null is {a >= null}");
Console.WriteLine($"{a} < null is {a < null}");
Console.WriteLine($"{a} == null is {a == null}");
// Output:
// 10 >= null is False
// 10 < null is False
// 10 == null is False
```

```
int? b = null;
int? c = null;
Console.WriteLine($"null >= null is {b >= c}");
Console.WriteLine($"null == null is {b == c}");
// Output:
// null >= null is False
// null == null is True
```



Null-supression operátor - !



- Lehetséges nullértékű esetben letiltható vele a kifejezés összes null értékű figyelmeztetése
 - Jelentése: „biztos vagyok benne, hogy ez nem lesz null”
 - Futásidőben az operátornak nincs hatása, viszont ha az érték mégis null, akkor **NullReferenceException**



```
public static void Main()
{
    Person? p = Find("John");
    if (IsValid(p))
    {
        Console.WriteLine($"Found {p!.Name}");
    }
}

public static bool IsValid(Person? person)
    => person is not null && person.Name is not null;
```




Null feltételes taghozzáférés - ?. és ?[]



- Tag- vagy elemhozzáférési művelet végrehajtása csak akkor, ha az operandus nem null értékű
- Ellenkező esetben a függvény null-t ad vissza



```
int GetSumOfFirstTwoOrDefault(int[] numbers)
{
    if ((numbers?.Length ?? 0) < 2)
    {
        return 0;
    }
    return numbers[0] + numbers[1];
}
```

```
Console.WriteLine(GetSumOfFirstTwoOrDefault(null)); // output: 0
Console.WriteLine(GetSumOfFirstTwoOrDefault(new int[0])); // output: 0
Console.WriteLine(GetSumOfFirstTwoOrDefault(new[] { 3, 4, 5 })); // output: 7
```





?. és ?[] példa



```
double SumNumbers(List<double[]> setsOfNumbers, int indexOfSetToSum)
{
    return setsOfNumbers?[indexOfSetToSum]?.Sum() ?? double.NaN;
}
```

```
var sum1 = SumNumbers(null, 0);
Console.WriteLine(sum1); // output: NaN
```

```
var numberSets = new List<double[]>
{
    new[] { 1.0, 2.0, 3.0 },
    null
};
```

```
var sum2 = SumNumbers(numberSets, 0);
Console.WriteLine(sum2); // output: 6
```

```
var sum3 = SumNumbers(numberSets, 1);
Console.WriteLine(sum3); // output: NaN
```

