



# Nullable

Répásné Babucs Hajnalka

Répás Csaba



# 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
```

