

C# Újdonságok

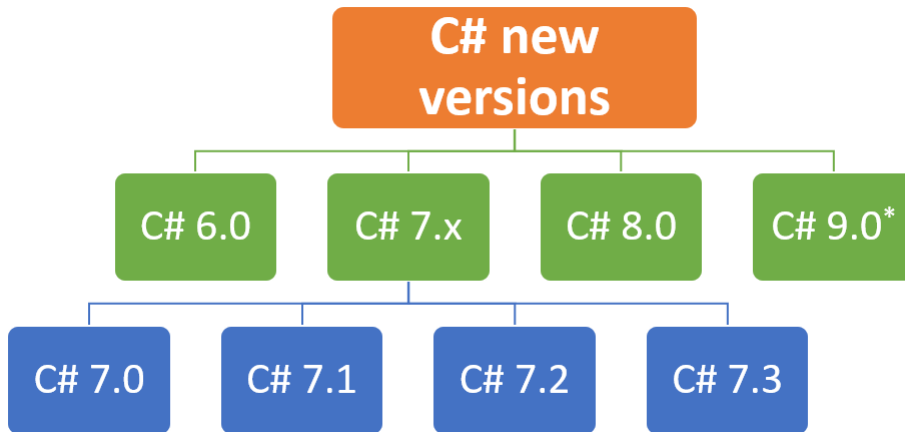
Jánosi-Rancz Katalin Tünde

Sapientia EMTE
tsuto@ms.sapientia.ro

A C# fejlődése.

Evolution of C#		(dotnetcodetree)					
VS IDE Version	Key features						
C# 1.0	VS2002	Managed Code					
C# 2.0	VS2005	Partial Class	Generics	Anonymous Methods	Nullable Types	Covariance Contra-variance	Lambda Expression
C# 3.0	VS2008	Anonymous Types	Extension Method	Lambda Expression	Implicit type (var)	LINQ	Expression Tree
C# 4.0	VS2010	Named Arguments	Late Binding	Optional Parameters	More COM Support		
C# 5.0	VS2012	Caller information	Async programming				
C# 6.0	VS2015	String Interpolation	Expression Bodied methods	await in catch and finally block	Exception Filters	null conditional operator	auto property initializer
C# 7.0	VS2017	Tuples	Pattern Matching	Out Variables	Local functions	ref return and ref local	Throw Expression

C# 6.0: open source compiler, szintaxis egyszerűsítés



C# 7.0 újdonság- Tuple type

- ▶ Létrehozhatunk tuple-okat úgy, hogy mindenik taghoz hozzárendelünk egy értéket, és opcionálisan szemantikus neveket adhatunk a tagoknak:

```
//szintaxis:  
(<datatype> [name1],<datatype> [name2]) methodName(parameters){  
    return (<val1>,<val2>);}  
</val2></val1></datatype></datatype>
```

```
(string Alpha, string Beta) namedLetters = ("a", "b");  
Console.WriteLine($"{namedLetters.Alpha}, {namedLetters.Beta}");
```

- ▶ megadhatjuk a mezők nevét a jobb oldalon is:

```
var alphabetStart = (Alpha: "a", Beta: "b");  
Console.WriteLine($"{alphabetStart.Alpha}, {alphabetStart.Beta}");
```

- ▶ tuple-ek kicsomagolása

```
(int max, int min) = Range(numbers);  
Console.WriteLine(max);  
Console.WriteLine(min);
```

Mit hozott a .NET 5.0?

- ▶ Egyetlen fájlba csomagolt alkalmazás

```
dotnet publish -r linux-x64 --self-contained true /p:  
  PublishSingleFile=true
```

- ▶ ARM64 támogatás

- ▶ A DLL fájl tartalmazhatja a .pdb-t is.

- ▶ Null-ozható lett még több referencia típus.

- ▶ Kovariáns visszatérés – Override-olt tulajdonság specifikusabb típust tud visszaadni mint az őosztály.

- ▶ F sharp 5.0

- ▶ C# 9.0

- ▶ új kulcsszavak: `and`, `or`, `not`, `with`

- ▶ Record típus

- ▶ init előtag setter-ekhez (init only setter)

- ▶ Legfőbb szintű állítások (Top-level statements)

2. Mit hozott a .NET 5.0?

- ▶ Performancia javítások
 - ▶ HTTP 1.1 és HTTP/2 implementáció
 - ▶ Optimalizáció string műveletekre és reguláris kifejezésekre
 - ▶ Garbage Collector optimalizáció
 - ▶ a .NET 5.0 szerver teljesítménye 60% -kal gyorsabb, mint a .NET Core 3.1.
 - ▶ a .NET 5.0 kliens teljesítménye 230% -kal gyorsabb, mint a .NET Core 3.1
 - ▶ JSON szerializáció 42%-al gyorsabb lett

Top-level statements

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

Helyett:

```
System.Console.WriteLine("Hello World!");
```

Record típus, C# 9.0

A rekordtípus lényege, hogy könnyebb módot kínál megváltoztathatatlan jellemzők használatára, és jobb egyenlőség vizsgálatot kínál

- ▶ referencia típus; immutable (megváltoztathatatlan), mint a `string`

```
public record Person
{
    public string LastName { get; }
    public string FirstName { get; }

    public Person(string first, string last) => (FirstName, LastName)
        = (first, last);
}
```

- ▶ létrehoz egy `Person` típust, amely két read only tulajdonságot tartalmaz. Ha megnézed az IL-t, ez egy *osztály*.
- ▶ megváltoztathatatlan abban, hogy a létrehozás után egyik tulajdonsága sem módosítható.
- ▶ használható rá:
 - ▶ a `ToString()`
 - ▶ érték alapú egyenlőség vizsgálat
 - ▶ tagok másolata és klónozása

Recordok támogatják az öröklődést

- ▶ deklarálhatunk egy Personból származó új rekordot:

```
public record Teacher : Person
{
    public string Subject { get; }

    public Teacher(string first, string last, string sub) : base(first,
        last) => Subject = sub;
}
```

- ▶ sealed-é is tehetjük, hogy megakadályozzuk a további származtatást

```
public sealed record Student : Person
{
    public int Level { get; }

    public Student(string first, string last, int level) : base(first,
        last) => Level = level;
}
```

Recordok egyenlőség vizsgálata

- ▶ az egyenlőség vizsgálat érték alapú, ellenőrzi, hogy a típusok egyeznek-e.
- ▶ egy Student nem lehet = egy Person-al, még akkor sem, ha a két rekordban ugyanaz a név

```
var person = new Person("Bill", "Wagner");  
var student = new Student("Bill", "Wagner", 11);  
  
Console.WriteLine(student == person); // false
```

- ▶ A rekordok következetes karakterlánc-reprezentációt generálnak
- ▶ ToString() a következő stringet téríti vissza

```
"Student { LastName = Wagner, FirstName = Bill, Level = 11 }"
```

Positional records - Pozíciórekordok

- ▶ az eddig bemutatott példák a hagyományos szintaxist használják a tulajdonságok deklarációjához.
- ▶ Positional records- tömörebb forma

```
public record Person(string FirstName, string LastName);  
  
public record Teacher(string FirstName, string LastName, string Subject)  
    : Person(FirstName, LastName);  
  
public sealed record Student(string FirstName, string LastName, int Level)  
    : Person(FirstName, LastName);
```

- ▶ *NB*: {} helyett ; végződnek, mivel ezek a rekordok nem tartalmaznak további törzset, metódusokat.

► további metódusokat is felvehet

```
public record Pet(string Name)
{
    public void ShredTheFurniture() => Console.WriteLine("Shredding
        furniture");
}

public record Dog(string Name) : Pet(Name)
{
    public void WagTail() => Console.WriteLine("It's tail wagging time");

    public override string ToString()
    {
        StringBuilder s = new();
        base.PrintMembers(s);
        return $"{s.ToString()} is a dog";
    }
}
```

Deconstruct metódus

- ▶ a fordító elkészíti a Deconstruct metódust a pozíciórekordokhoz
- ▶ a Deconstruct metódus olyan paraméterekkel rendelkezik, amelyek megegyeznek a rekordtípusban szereplő összes nyilvános tulajdonság nevével
- ▶ a Deconstruct metódus használható a rekord dekomponálására a komponens tulajdonságaiba:

```
var person = new Person("Bill", "Wagner");  
  
var (first, last) = person;  
Console.WriteLine(first);  
Console.WriteLine(last);
```

- ▶ a `with` kifejezés utasítja a fordítót egy rekord másolatának létrehozására, de megadott tulajdonságok módosításával:

```
Person brother = person with { FirstName = "Paul" };
```

- ▶ egy pontos másolatot klónozhatunk

```
Person clone = person with { };
```

Properties - Init only setters

- ▶ az objektum tagjainak inicializálására használjuk
- ▶ bármilyen osztályhoz vagy struktúrához használhatók
- ▶ az érték létrehozáskor inicializálódik, utána csak olvashatóak

```
public struct WeatherObservation
{
    public DateTime RecordedAt { get; init; }
    public decimal TemperatureInCelsius { get; init; }

    public override string ToString() =>
        $"At {RecordedAt:h:mm tt} on {RecordedAt:M/d/yyyy}: " +
        $"Temp = {TemperatureInCelsius}";
}
```

```
var now = new WeatherObservation
{
    RecordedAt = DateTime.Now,
    TemperatureInCelsius = 20,
};
```

- ▶ megőrzik a változtathatlanságot, módosítani már nem lehet

```
now.TemperatureInCelsius = 18; // Error! CS8852.
```

Mintaillesztési fejlesztések

- ▶ új kulcsszavak: `and`, `or`, `not`

```
public static bool IsLetter(this char c) => c is >= 'a' and <= 'z' or
    >= 'A' and <= 'Z';
public static bool IsLetterOrSeparator(this char c) => c is (>= 'a'
    and <= 'z') or (>= 'A' and <= 'Z') or '.' or ',';
```

- ▶ új szintaxis null ellenőrzéshez

```
if (e is not null)
{
    // ...
}
```

```
if (pollingType is null or { Length: >15 })
{
    return null;
}
```


- ▶ logikai `and` pattern egy új szintaxisú switch-ben

```
public static byte GetRxBufferNumber(Address address) =>
address switch
{
    >= Address.RxB0D0 and <= Address.RxB0D7 => 0,
    >= Address.RxB1D0 and <= Address.RxB1D7 => 1,
    _ => throw new ArgumentException(nameof(address), $"Invalid
        address value {address}.");
};
```

Fit és finish funkciók - a `new` utáni típus elhagyása

- ▶ elhagyható a típus egy új kifejezésben, ha a létrehozott objektum típusa már ismert

```
private List<WeatherObservation> _observations = new();
```

- ▶ vegyünk egy `ForecastFor ()` metódust a következő szignatúrára:

```
public WeatherForecast ForecastFor(DateTime forecastDate,  
    WeatherForecastOptions options)
```

- ▶ meghívhatod a következőképpen:

```
var forecast = station.ForecastFor(DateTime.Now.AddDays(2), new());
```

- ▶ visszatéríthetjük az alapértelmezett konstruktor által létrehozott példányt a `return new();` utasítással

Továbbfejlesztett Dictionary <K, V> támogatás

- ▶ a JsonSerializer most már *nem* karakterláncos kulcsokkal is támogatja a Dictionary-t

```
using System;
using System.Collections.Generic;
using System.Text.Json;

Dictionary<int, string> numbers = new ()
{
    {0, "zero"},
    {1, "one"},
    {2, "two"},
    {34, "thirty four"},
    {55, "fifty five"},
};

var json = JsonSerializer.Serialize<Dictionary<int, string>>(numbers);
Console.WriteLine(json); //{"0":"zero","1":"one","2":"two","34":"
    thirty four","55":"fifty five"}

var dictionary = JsonSerializer.Deserialize<Dictionary<int, string>>(
    json);
Console.WriteLine(dictionary[55]); //fifty five
```

Nullozható típusok

- ▶ C# 2-től lehet egy érték típusnak null értéket adni

```
Nullable<int> x= null;  
//vagy rövidebben  
int? x = null;
```

- ▶ C# 8 előtt minden referenciatípus nullozható volt

```
WriteLine(customer.Id); //ha null, System.NullReferenceException  
WriteLine(customer?.Id); //nincs hiba
```

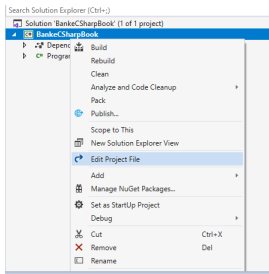
- ▶ Null-conditional Operator (?.) a Null-Coalescing operator(??) -al

```
WriteLine($"customer name:{customer?.Name?? "not provided or null"}")  
;  
WriteLine($"Name Length:{customer?.Name?.Length?? 0}");
```

- ▶ C# 8-től jelezhetjük, hogy szeretnénk ha egy referencia típus nem nullozható lenne

```
class Person
{
    string FirstName;    // NEM lehet null
    string? MiddleName; // Lehet null
    string LastName;    // NEM lehet null
}
```

Nem nullozható típusok engedélyezése



```
BankeCSharpBook.csproj Program.cs
1 <Project Sdk="Microsoft.NET.Sdk">
2
3   <PropertyGroup>
4     <OutputType>Exe</OutputType>
5     <TargetFramework>netcoreapp3.0</TargetFramework>
6     <Nullable>enable</Nullable>
7   </PropertyGroup>
8
9 </Project>
```

C# 9 Egyszerűsített null paraméter érvénytelenítés

-Null-forgiving operator (!)

- ▶ ! prefixként logikai tagadó operátor
- ▶ ! postfixként Null-forgiving (bang) operátor

```
// C# 8
void AddCustomer(Customer customer)
{
    if (customer is null)
    {
        throw new ArgumentNullException(nameof(customer));
    }
    _customers.Add(customer);
}
```

```
// C# 9
void AddCustomer(Customer customer!) => _customers.Add(customer);
```

Régi-új Tool - Winform

The screenshot shows a Visual Studio environment with a WinForms application in design mode. The application window has a blue header with the text "Cars" and a list of categories: "Reservations", "Inspections", and "Employees". A large image of a dark blue car is displayed. Below the image is a form with input fields for "Make", "Model", "Year", "VIN Number", "Engine", "Horsepower", "Wheels", "Tires", "Transmission", and "Lights", along with an "Add Car" button. The Visual Studio interface includes the Toolbox, Solution Explorer, Properties window, and Error List.

Toolbox: Pointer, Button, CheckBox, CheckedListBox, ComboBox, DateTimePicker, Label, LinkLabel, ListBox, ListView, MaskedTextBox, MonthCalendar, NotifyIcon, NumericUpDown, PictureBox, ProgressBar, RadioButton, RichTextBox, TextBox, ToolTip, TreeView, Containers (.NET Core), FlowLayoutPanel, GroupBox, Panel, SplitContainer, TabControl, TabLayoutPanel, Menus & Toolbars (.NET Core), ContextMenuStrip, MenuStrip, StatusStrip, ToolStrip, Components (.NET Core).

Solution Explorer: WindowsFormsApp, Properties, Resources.resx, Resources, car.PNG, MainForm.cs, MainForm.Designer.cs, MainForm.resx, MainForm, Program.cs.

Properties: Name: MainForm, AcceptButton: (none), AccessibleDes: AccessibleNav, AccessibleRole: Default, AllowDrop: False, AutoScaleMode: Font, AutoScroll: False, AutoScrollMargin: 0, 0, AutoScrollMin: False, AutoSizeMode: GrowOnly, AutoValidate: EnablePreventFocus, BackColor: (none), BackgroundImage: (none), BackgroundImageTile: CancelButton: (none), CausesValidation: True, ContextMenuStrip: (none), DataBindings: The data bindings for the control.

Error List: 0 Errors, 0 Warnings, 0 of 1 Message. Build - IntelliSense.

Többé nem támogatott technológiák

- ▶ Web Forms
- ▶ Windows Communication Foundation (WCF)
- ▶ Windows Workflow Foundation