

Jazyk C# a .NET

prednáška

2



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Základy jazyka C# (1)

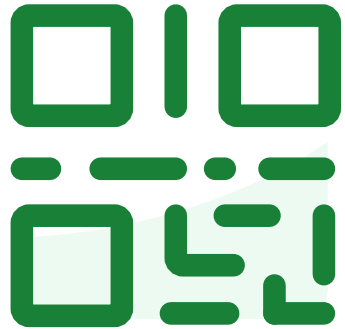
Dátové typy a triedy

Ing. **Štefan Toth**, PhD.

27.02.2025

slido

Please download and install the Slido app on all computers you use



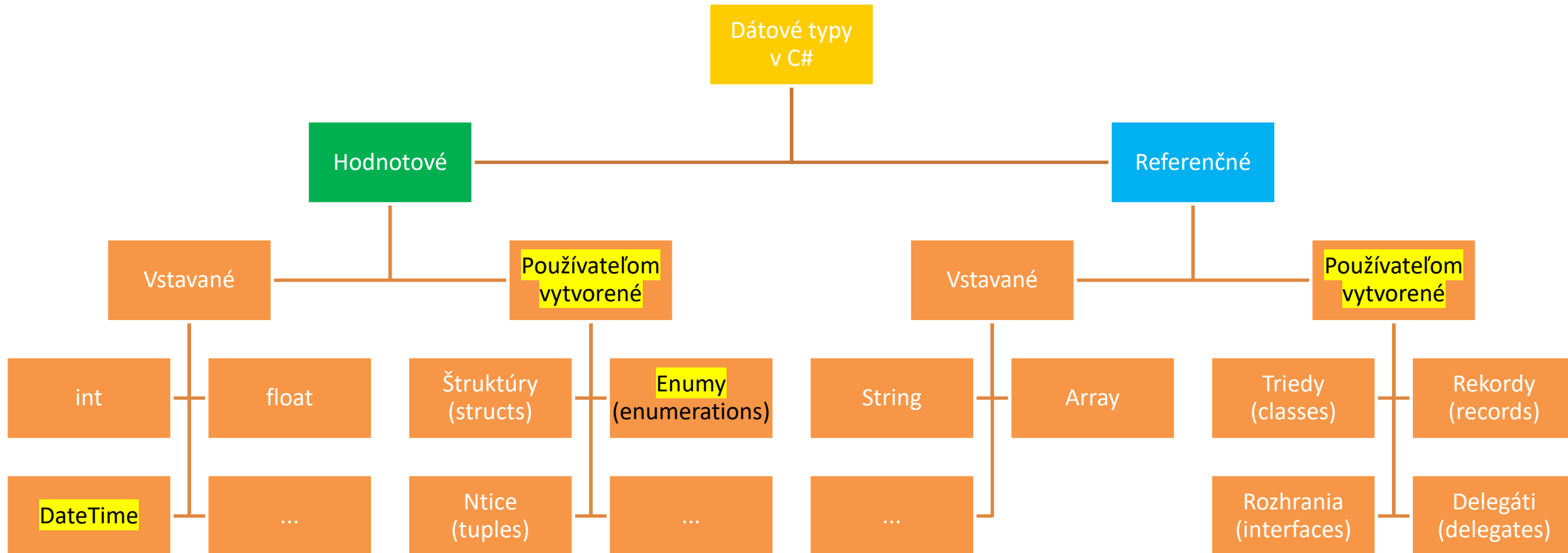
**Join at slido.com
#3237110**

① Start presenting to display the joining instructions on this slide.

Obsah

- **Dátové typy** (data types)
- **Triedy** (classes)

Dátové typy (data types) v C#



Hodnotové typy (value types)

- **Jednoduché vstavané typy (štruktúry)**
 - Celočíselné so znamienkom: **sbyte, short, int, long, nint**
 - Celočíselné bez znamienka: **byte, ushort, uint, ulong, nuint**
 - Čísla s pohyblivou rádovou čiarkou: **float, double** (podľa štandardu IEEE 754)
 - Desatinné čísla s vysokou presnosťou: **decimal**
 - Unicode (UTF-16) znaky: **char**
 - Boolovské: **bool**
- **Štruktúry**
 - Používateľsky definované typy v tvare **struct S { ... }** alebo **ref struct S { ... }** / **readonly ref struct S { ... }** (výhradne na zásobníku) alebo **record struct S { ... }**
- **Vymenované typy**
 - Používateľsky definované typy v tvare **enum E { ... }**
- **Nullable typy**
 - Rozšírenie všetkých ostatných hodnotových typov o null hodnotu – typ **Nullable<T>** alebo skrátené s použitím otáznika **T?**
- **Tuple typy**
 - N-tice hodnôt v tvare **(p1, p2, ..., pN)**

Referenčné typy (reference types)

- **Triedy** (class)
 - Vstavané typy v jazyku C#:
 - **object** (alias pre **System.Object**) – základná trieda pre všetky ostatné typy
 - **string** (alias pre **System.String**) – reťazec (v UTF-16)
 - **dynamic** – dynamický typ za behu, ktorý odsúva kontrolu v čase kompilácie
 - Používateľsky definované typy v tvare **class C { ... }**
- **Rozhrania** (interface)
 - Používateľsky definované typy v tvare **interface I { ... }**
- **Delegáti** (delegate)
 - Používateľsky definované typy v tvare **delegate typ D(...)**
- **Rekordy** (record)
 - Poskytuje vstavané funkcionality pre dáta: **record R();** alebo **record R { ... }**
- **Polia**
 - Jednorozmerné a viacrozmerné polia (napr. **int[], int[,]** alebo **int[][] ...**)

Číselné dátové typy v C#/.NET – detailnejšie

Kategória	Počet bitov	Typ (C#)	Typ (.NET)	Rozsah/presnosť
Celé čísla so znamienkom (záporné a kladné čísla)	8	sbyte	System.SByte	-128 ... 127
	16	short	System.Int16	-32 768 ... 32 767
	32	int	System.Int32	-2 147 483 648 ... 2 147 483 647
	64	long	System.Int64	-9 223 372 036 854 775 808 ... 9 223 372 036 854 775 807
	32 alebo 64	nint	System.IntPtr	Závisí od platformy (v runtime)
Celé čísla bez znamienka (iba kladné čísla)	8	byte	System.Byte	0 ... 255 ($0 - 2^8$)
	16	ushort	System.UInt16	0 ... 65 535 ($0 - 2^{16}$)
	32	uint	System.UInt32	0 ... 4 294 967 295 ($0 - 2^{32}$)
	64	ulong	System.UInt64	0 ... 18 446 744 073 709 551 615 ($0 - 2^{64}$)
	32 alebo 64	nuint	System.UIntPtr	Závisí od platformy (v runtime)
Čísla s pohyblivou rádovou čiarkou	16	float	System.Half	-65 504 ... +65 504, (10 bitov mantisa, 5 bitov exponent, 1 bit znamienko)
	32		System.Single	$-3,402823 \times 10^{38} \dots 3,402823 \times 10^{38}$, presnosť ~6-9 číslic (23 bitov mantisa, 8 bitov exponent, 1 bit znamienko)
	64		System.Double	$-1,79769313486232 \times 10^{308} \dots 1,79769313486232 \times 10^{308}$, presnosť ~15-17 číslic (52 bitov mantisa, 11 bitov exponent, 1 bit znamienko)
Desatinné čísla	128	decimal	System.Decimal	$\pm 1,0 \times 10^{-28} \dots \pm 7,9228 \times 10^{28}$, 28-29-číslicová presnosť

- Najmenšie a najväčšie možné číslo môžeme zistiť pomocou konštánt **MinValue** a **MaxValue** v každom type (napr. **int.MinValue** alebo **Int32.MinValue**)

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Premenné (1)

- **Premenná** má typ a hodnotu, ktorá sa časom mení, môžeme ju deklarovať klasicky alebo aj inicializovať na určitú hodnotu:
 - Typ identifikátor;
 - Typ identifikátor = hodnota;
- C# 3.0 zaviedol kľúčové slovo **var** s odvodením typu (type inference), kedy sa **typ** premennej **odvodí z pravej strany** v čase kompilácie:
 - **var** identifikátor = new Typ();
 - **var** identifikátor = new Typ(parameter1, parameter2, parameterN);
- C# 9.0 zaviedol ďalšiu možnú syntax s kľúčovým slovom **new** bez použitia typu (target-typed new expression) – **ak je typ známy na ľavej strane**, môžeme ho za kľúčovým slovom **new** vynechať:
 - Typ identifikátor = **new()**; // Volá sa bezparametrický konštruktor
 - Typ identifikátor = **new**(parameter1, parameter2); // Konštruktor s
// dvomi parametrami

Premenné (2) – príklady

```
int i1 = 123; // Deklarácia a inicializácia
var i2 = 123; // Odvodenie typu v čase kompilácie („type inference“)

string s1 = "Hello, World!";
var s2 = "Hello, World!";

// Nasledujúce príkazy sú možné, avšak veľmi sa nepoužívajú:
var s3 = new string("Hello, World!");
string s4 = new("Hello, World!");
```

```
StringBuilder sb1 = new StringBuilder("Jazyk C# a .NET");
var sb2 = new StringBuilder("Jazyk C# a .NET");
StringBuilder sb3 = new("Jazyk C# a .NET");
```

Kľúčové slová (keywords)

- Príklady:
 - `if, else, for, foreach, while, do, int, string, interface, class, new, ...`
- Nie je možné ich použiť ako identifikátory v programe – ak by sme napriek tomu požadovali, zvykne sa používať predpona `@`, napr.:
 - `int @new = 123;`
- Zoznam všetkých kľúčových slov:
 - <https://learn.microsoft.com/sk-sk/dotnet/csharp/language-reference/keywords/>

Rozsah platností premenných (variable scope)

- **Rozsah platnosti** (scope) premennej je oblasť kódu, v ktorej je táto premennej prístupná
 - Dátový člen triedy je v rozsahu celej triedy
 - Lokálna premenná v metóde je v rozsahu platnosti až do uzatváracej zátvorky na konci bloku, v ktorej je deklarovaná
 - Lokálna premenná v cykle (for/while) je v rozsahu tela cyklu
- Poznámka: kompilátor premiestňuje všetky deklarácie premenných na začiatok rozsahu platnosti

```
void Method()
{
    for (int i = 1; i <= 5; i++)
    {
        int month = 2; // Chyba CS0136
        Console.WriteLine(month);
    }

    int month = 6;
    Console.WriteLine(month);
}
```

Konštanty

- **Hodnota** konštanty **je nemenná**
- **Kompilátor nahrádza** v kóde použitie konštanty jej **hodnotou**
- Vlastnosti:
 - Inicializované v mieste deklarácie
 - Hodnota známa v dobe prekladu
 - Vždy sú statické, hoci sa kľúčové slovo static nepoužíva

```
const double Pi = 3.1415926535897932384626433832795028841971693993751058209749445923;
```

```
const int Width = 1920;  
const int Height = 1080;  
const int Resolution = Width * Height;
```

```
int Width = 1920;  
int Height = 1080;  
const int Resolution = Width * Height; // Chyba  
// CS0120
```

Celočíselné literály

- Celé čísla môžu byť zadané v tvare:
 - **desiatkovom** (decimálnom): bez akejkoľvek predpony (prefixu)
 - **šestnástkovom** (hexadecimálnom): s predponou **0x** alebo **0X**
 - **dvojkovom** (binárne): s predponou **0b** alebo **0B** (dostupné od C# 7.0)
- Na oddelenie cifier pre lepšiu čitateľnosť môžeme použiť aj znak **oddelovača číslic** (digit separator) „_“, kompilátor ho ignoruje
- Príklady:
 - `var decimalLiteral = 42;`
 - `var hexLiteral = 0x2A;`
 - `var binaryLiteral = 0b_0010_1010;`

Prípony k číselným literálom

- **U** / **u** = unsigned int
- **L** / **l** = long
- **UL** / **ul** / iné kombinácie U a L = unsigned long
- **F** / **f** = float
- **D** / **d** = double
- **M** / **m** = decimal (M / m ako „money“)

Odvodenie celočíselného typu

- **Ak literál nemá príponu**, jeho typ je prvý z nasledujúcich typov, v ktorých je možné jeho hodnotu vyjadriť: **int**, **uint**, **long**, **ulong**
- **Ak literál končí na príponu U alebo u**, jeho typ je prvý z nasledujúcich typov, v ktorých môže byť jeho hodnota zastúpená: **uint**, **ulong**
- **Ak literál končí na príponu L alebo l**, jeho typ je prvý z nasledujúcich typov, v ktorých môže byť jeho hodnota zastúpená: **long**, **ulong**
- **Ak literál končí na prípony UL, Ul, uL, ul, LU, Lu, lU alebo lu**, jeho typ bude **ulong**
 - **Odporúča sa používať veľké písmená** (pri malom písmen L = l môže byť zámena s číslom 1)

Zdroj: <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types>

Implicitné číselné konverzie

- **Implicitné** – prebiehajú automaticky, nedochádza k strate informácií (výnimkou môže byť float/double)

Z typu	Na typ
sbyte	short, int, long, float, double, decimal alebo nint
byte	short, ushort, int, uint, long, ulong, float, double, decimal, nint alebo nuint
short	int, long, float, double, decimal alebo nint
ushort	int, uint, long, ulong, float, double alebo decimal, nint alebo nuint
int	long, float, double alebo decimal, nint
uint	long, ulong, float, double, decimal alebo nuint
long	float, double alebo decimal
ulong	float, double alebo decimal
float	double
nint	long, float, double alebo decimal
nuint	ulong, float, double alebo decimal

```
// Implicitná konverzia z int na byte:  
byte a = 13;  
  
byte b = 300; // Chyba CS0031:  
Constant value '300' cannot be  
converted to a 'byte'
```

Zdroj: <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/numeric-conversions>

Explicitné číselné konverzie

- Explicitné** – vyžadujú manuálne pretypovanie (casting), pretože môže dôjsť k strate informácií

Z typu	Na typ
sbyte	byte, ushort, uint, ulong alebo nuint
byte	sbyte
short	sbyte, byte, ushort, uint, ulong alebo nuint
ushort	sbyte, byte alebo short
int	sbyte, byte, short, ushort, uint, ulong alebo nuint
uint	sbyte, byte, short, ushort, int alebo nint
long	sbyte, byte, short, ushort, int, uint, ulong, nint alebo nuint
ulong	sbyte, byte, short, ushort, int, uint, long, nint alebo nuint
float	sbyte, byte, short, ushort, int, uint, long, ulong, decimal, nint alebo nuint
double	sbyte, byte, short, ushort, int, uint, long, ulong, float, decimal, nint alebo nuint
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, float, double, nint alebo nuint
nint	sbyte, byte, short, ushort, int, uint, ulong alebo nuint
nuint	sbyte, byte, short, ushort, int, uint, long alebo nint

```
double x = 123.456;  
int y = (int)x; // Explicitná konverzia
```


Zdroj: <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/numeric-conversions>

Príklady


```
int a = 123;    // System.Int32
Int32 b = 123; // System.Int32 = int
var c = 123;    // System.Int32 = int

var uintNumber = 123U; // System.UInt32 = uint
var uintNumber2 = 4_294_967_295; // System.UInt32 = uint
```

```
int a = 123;
```

 **readonly struct** System.Int32
Represents a 32-bit signed integer.

```
var c = 123;
```

 **readonly struct** System.Int32
Represents a 32-bit signed integer.

```
uint binary = 0b_1111_1110_1101_1100_1010_1001_1000;
long number = 0x_1234_5678_9abc;

byte a = 17;
byte b = 300; // CS0031: Constant value '300' cannot be converted to a 'byte'

var signedByte = (sbyte)42; // System.SByte = sbyte
var longVariable = (long)42; // System.Int64 = long
```


slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.



Aký typ je priradený premennej deklarovanej s 'var'?

① Start presenting to display the poll results on this slide.



Aký je hlavný rozdiel medzi implicitným a explicitným pretypovaním?

① Start presenting to display the poll results on this slide.

Trieda (class) (1)

- Triedy sú **referenčné typy** vytvárané na **halde** (heap)
- Deklarácia triedy obsahuje:
 - **prístupový modifikátor** (access modifier) – **public** alebo **internal**
 - Ak nie je modifikátor uvedený (vynechaný), implicitne je **internal** (prístup iba v rámci zostavenia, t. j. projektu)
 - Ak je ale trieda **vnorená** (deklarovaná napr. v inej triede), potom táto môže mať aj **ďalšie prístupové modifikátory** (**private**, **protected**, **private protected**, **protected internal**, ...)
 - kľúčové slovo **class**
 - **identifikátor** (identifier) – názov triedy
 - **telo triedy** (class body) – obsahuje členy triedy (class members)

```
// Príklad triedy MyClass
public class MyClass
{
    // Telo triedy
}
```

```
// Príklad triedy MyClass bez uvedenia modifikátora (internal )
class MyInternalClass
{
    // Telo triedy
}
```


Trieda (2)

- **Trieda** obsahuje rôznych **členov** (members), ktoré môžu byť:
 - **statické** (static) – patria triede
 - **inštančné** (instance) – patria objektu, t. j. inštancii triedy

```
class MyClass
{
    private static int s_staticField = 0;
    private int _instanceField = 0;

    public static void StaticMethod()
    {
        Console.WriteLine("Toto je statická metóda.");
    }

    public void InstanceMethod()
    {
        Console.WriteLine("Toto je inštančná metóda.");
    }
}
```

Trieda (3)

- **Inštancia** (instance) triedy sa vytvára s použitím operátora **new**, ktorý alokuje pamäť, volá konštruktor na inicializovanie inštancie a vracia referenciu na túto inštanciu:

```
var myInstance = new MyClass();  
MyClass myInstance2 = new();  
var list = new List<MyClass>();
```

- Okrem toho je možné použiť aj **inicializátor objektu** (object initializer) za konštruktorom:

```
// Vytvorenie inštancie spolu s inicializáciou objektu pomocou zložených zátvoriek {}. V prípade  
// bezparametrického konštruktora sú dokonca guľaté zátvorky volania konštruktora nepovinné  
var list = new List<MyClass>()  
{  
    new MyClass(),  
    new MyClass(), // Aj posledná čiarka je nepovinná  
};
```

Členy triedy (class members)

- **Dátové členy** (fields) a **dátové členy len na čítanie** (readonly fields)
- **Konštanty** (constants)
- **Vlastnosti** (properties)
- **Metódy** (methods)
- **Konštruktory** (constructors)
- **Finalizéry** (finalizers) – v minulosti označované ako deštruktory (destructors)
- **Dekonštruktory** (deconstructors)
- **Indexery** (indexers)
- **Udalosti** (events)
- **Operátory** (operators)
- **Vnorené typy** (nested types)

Dátový člen (fields)

- Dátové členy – „fields“ – **premenné deklarované v rozsahu triedy**
- Pozor: z hľadiska **objektovo orientovaného programovania** sa používa pojem **atribút**, v C# sa však termín **atribút (attribute)** používa na úplne iný účel – na pridávanie metadát do kódu (ekvivalent – anotácia v Java)
- Dobrá prax v OOP: používať **private**
- **Konvencia pomenovania** – prvé malé písmeno (napr. firstName) alebo podčiarkovník, ak je private (_firstName)
- Ak sa modifikátor prístupu pri „fielde“ neuvedie (napr. _lastName v ukážke), implicitne je **private** (pre čitateľnosť kódu je lepšie modifikátor prístupu explicitne použiť)

```
public class Person
{
    // ...
    private string _firstName;
    string _lastName;
    // ...
}
```

Modifikátor static

- Zdieľa dátový člen (field) so všetkými inštanciami triedy
- Konvencia pomenovania – prvé malé písmeno s alebo t (ak je použitý atribút ThreadStatic) s **podčiarkovníkom _**, ak sa jedná o **private static** alebo **internal static** člen:

```
public class PeopleFactory
{
    //...
    private static int s_peopleCount;

    [ThreadStatic] static double t_sum = 0.0;
    //...
}
```

Dátový člen len na čítanie (readonly field)

- Iba na čítanie, hodnotu je možné zmeniť iba v konštruktoroch alebo v mieste deklarácie
- Po vytvorení inštancie už nie je možné hodnotu zmeniť:

```
public class Person
{
    // ...
    public Person(string firstName, string lastName)
    {
        _firstName = firstName;
        _lastName = lastName;
    }

    private readonly string _firstName;
    private readonly string _lastName;
    // ...
}
```

Vlastnosti (properties) (1)

- Namiesto get a set metód má C# špeciálnu syntax pre **vlastnosti** (properties) obsahujúce **get** a **set** časť, pričom v set časti je k dispozícii hodnota **value**

```
public class Person
{
    // Field:
    private int _age;

    // Starší typ zápisu property:
    public int Age
    {
        get { return _age; }
        set { _age = value; }
    }
}
```

```
public class Person
{
    // Field:
    private int _age;

    // Novší typ zápisu property:
    public int Age
    {
        get => _age;
        set => _age = value;
    }
}
```

```
// Použitie vlastnosti
var person = new Person();

// Volá sa get časť vlastnosti
Console.WriteLine(person.Age);

// Volá sa set časť vlastnosti
person.Age = 25;

// Volá sa get časť:
Console.WriteLine(person.Age);
```

0
25

- Na vygenerovanie kostry kódu je vo Visual Studiu skratka **propfull**

Vlastnosti (2) – automatické vlastnosti

- **Automaticky implementované vlastnosti** (auto-implemented properties) – **kompilátor** automaticky **vytvorí dátový člen** („field“) spolu s **get** a **set** časťami, ktoré vracajú hodnotu a zapisujú hodnotu do tohto člena:

```
public class Person
{
    //...
    public int Age { get; set; }
    //...
}
```

```
public class Person
{
    // Môžeme aj inicializovať vlastnosť
    public int Age { get; set; } = 25
    //...
}
```

- Na vygenerovanie kostry kódu je vo Visual Studiu skratka **prop**

Vlastnosti (3) – vlastnosti len na čítanie

- **Vlastnosti len na čítanie** (read-only properties):

```
private readonly string _firstName;  
  
public string FirstName  
{  
    get => _firstName; // Chýba tu set časť, je dostupná iba get časť  
}
```

Vlastnosti (4) – vlastnosti len na čítanie – výraz

- **Vlastnosti** vyjadrené **ako výraz** (expression-bodied properties) pomocou operátora **=>** sú len na čítanie:

```
private readonly string _firstName;  
public string FirstName => _firstName; // Toto je vlastnosť  
  
public string FullName => $"{FirstName} {LastName}"; // Toto je vlastnosť
```

- **Pozor:** nepomýľte si vlastnosť s **metódou**, ktorú je takisto možné vyjadriť ako výraz s operátorom **=>**
– vlastnosť sa deklaruje bez zátvoriek, metóda má **guľaté zátvorky**:

```
public string FullName() => $"{FirstName} {LastName}"; // Toto je metóda
```

Vlastnosti (5) – vlastnosti len na zápis – nepoužívať

- **Vlastnosti len na zápis** (write-only properties) – aj keď ich jazyk C# umožňuje vytvoriť, považuje sa to za zlú programátorskú prax:

```
private string _someValue;  
public string SomeValue  
{  
    set => _someValue = value;  
}
```

- Ak potrebujete len zapisovať, odporúča sa radšej vytvoriť metódu:

```
private string _someValue;  
public string SetSomeValue(string value)  
{  
    _someValue = value;  
}
```

Vlastnosti (6) – prístupový modifikátor

- Môžeme zmeniť **prístupový modifikátor set** alebo **get** časti:

```
public int Age { get; private set; }
```

```
private string _firstName;  
public string FirstName  
{  
    private set => _firstName = value;  
}
```

Vlastnosti (7) – init

- Od C# 9 je možné namiesto kľúčového slova `set` použiť aj kľúčové slovo **`init`** (init-only set accessor)
- **Hodnota vlastnosti môže byť nastavená iba v konštruktore alebo pri inicializácii objektu:**

```
public class Book
{
    public Book(string title)
    {
        Title = title;
    }

    public string Title { get; init; }
    public string? Publisher { get; init; }
}
```

```
// Vlastnosť Title nastavená v konštruktore
Book book = new("Professional C#")
{
    // Vlastnosť Publisher nastavená
    // v inicializátore objektu
    Publisher = "Wrox Press"
};
```

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

slido



**Aké sú správne pomenovania identifikátorov
privátnych dátových členov (fields):**

ⓘ Start presenting to display the poll results on this slide.

slido



**Ktoré z uvedených sú platné vlastnosti
(properties):**

ⓘ Start presenting to display the poll results on this slide.

slido



Čo platí pre kľúčové slovo init vo vlastnosti?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Metódy (methods) (1)

- Deklarácia metódy:
 - Prístupový modifikátor metódy (private, public, ...)
 - Voliteľný modifikátor (virtual, abstract, sealed, ...)
 - Návratový typ
 - Názov metódy
 - Zoznam parametrov
 - Telo metódy

```
public bool IsSquare(Rectangle rect)
{
    return (rect.Height == rect.Width);
}
```

```
public bool IsSquare(Rectangle rect) => (rect.Height == rect.Width);
```

Metódy (2) – príklady

```
public class Math
{
    public int Value { get; set; }
    public int GetSquare() => Value * Value;
    public static int GetSquareOf(int x) => x * x;
}
```

```
// Zavolanie statickej metódy:
int x = Math.GetSquareOf(5);
Console.WriteLine($"Druha mocnina cisla 5 je {x}");
```

```
// Vytvorenie inštancie objektu Math
// a zavolanie inštančných členov triedy
Math math = new();
math.Value = 30;
Console.WriteLine($"Hodnota vlastnosti {nameof(math.Value)} je: {math.Value}");
Console.WriteLine($"Druha mocnina cisla {math.Value} je {math.GetSquare()}");
```

Druha mocnina cisla 5 je 25
Hodnota vlastnosti Value je: 30
Druha mocnina cisla 30 je 900

Preťažovanie metódy (method overloading)

- Rovnaký názov metódy, ale s odlišným **počtom** alebo **typmi parametrov**

```
public static class Console
{
    public static void WriteLine() { /* ... */ }

    public static void WriteLine(string? value) { /* ... */ }

    public static void WriteLine(int value) { /* ... */ }

    public static void WriteLine(string format, params object?[]? arg) { /* ... */ }

    // ...
}
```

Pomenované argumenty (named arguments)

- Používajú sa **názvy parametrov** metód **oddelené znakom dvojbodky** spolu s **argumentom** (hodnotou)
- Akákoľvek metóda môže byť vyvolaná s použitím pomenovaných argumentov, kompilátor ju automaticky zmení na klasické volanie metódy bez použitia názvov parametrov

```
// Ak zavoláme metódu, na prvý pohľad nie je jasné, čo zadané čísla predstavujú:  
MoveAndResize(1, 2, 100, 50);
```

```
// Môžeme použiť pomenované argumenty:  
MoveAndResize(x: 1, y: 2, width: 100, height: 50);
```

```
// Môžeme zmeniť aj poradie parametrov:  
MoveAndResize(1, 2, height: 50, width: 100);
```

```
public void MoveAndResize(int x, int y, int width, int height)  
{  
    //...  
}
```

Voliteľné argumenty (optional arguments)

- **Parametre** môžu byť **voliteľné** – vtedy používajú **predvolenú (default) hodnotu**
- Kompilátor zmení volanie tak, aby sa použila predvolená hodnota (pozor: nezmení predvolenú hodnotu s novou verziou kódu, preto pozor na zmeny v knižnici)

```
public void TestMethod(int notOptionalNumber, int optionalNumber = 42)
{
    Console.WriteLine(optionalNumber + notOptionalNumber);
}
```

```
TestMethod(1);
TestMethod(1, 2);
```

Voliteľné parametre a pomenované argumenty

- Ak máme **viacero voliteľných parametrov** s prednastavenými hodnotami, môžeme **zmeniť hodnotu iba určitého parametra** pomocou **pomenovaného argumenta**

```
public void TestMethod(int n, int opt1 = 10, int opt2 = 20, int opt3 = 30)
{
    Console.WriteLine(n + opt1 + opt2 + opt3);
}
```

```
TestMethod(1);
TestMethod(1, 2, 3);
TestMethod(1, opt3: 4); // Argumenty pre parametre opt1 a opt2 nemusíme uvádzať
```


Variabilný počet argumentov – params (1)

- Metóda môže byť volaná s **variabilným počtom argumentov**, ak sa použije kľúčové slovo **params** s typom **poľa** v deklarácii metódy:

```
public static void UseParamsInt(params int[] list)
{
    for (int i = 0; i < list.Length; i++)
    {
        Console.Write(list[i] + " ");
    }
    Console.WriteLine();
}
```

```
// Môžeme zavolať metódu bez argumentu alebo s ľubovoľným počtom argumentov:
UseParamsInt();
UseParamsInt(1);
UseParamsInt(1, 2, 3, 4);

// Dokonca môžeme použiť aj pole:
int[] intArray = { 5, 6, 7, 8, 9 };
UseParamsInt(intArray);
```

Variabilný počet argumentov – params (2)

- Ak má metóda viac parametrov, kľúčové slovo **params** je možné **použiť iba raz** a musí byť deklarované **na poslednom parametri**
- Príklady použitia v .NETe:

```
int x = 1;  
int y = 2;
```

```
// Zavolá sa metóda WriteLine(string format, object? arg0, object? arg1)  
Console.WriteLine("[{0}, {1}]", x, y);
```

```
// Zavolá sa metóda WriteLine(string format, params object?[]? arg)  
Console.WriteLine("[{0}, {1}] + [{2}, {3}]", x, y, x+1, y+1);
```

```
// Zavolá sa metóda CreateInstance(Type elementType, int length1, int length2, int length3)  
Array threeDimArray = Array.CreateInstance(typeof(double), 5, 5, 5);
```

```
// Zavolá metódu CreateInstance(Type elementType, params int[] lengths)  
Array fourDimArray = Array.CreateInstance(typeof(double), 5, 5, 5, 5);
```

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Modifikátory in, out a ref v parametroch metódy

- **Parametre** deklarované v metóde **bez** modifikátorov **in**, **out** alebo **ref** sú **odovzdané hodnotou**. Ak sú ale použité, potom sa **odovzdávajú odkazom** a ich správanie je nasledovné:
 - **in** – metóda nemôže zmeniť hodnotu parametra, argument parametra musí byť priradený a inicializovaný,
 - **out** – metóda musí priradiť hodnotu do parametra, argument parametra nemusí byť priradený, pretože sa inicializuje v metóde,
 - **ref** – metóda môže zmeniť hodnotu parametra, argument parametra musí byť priradený a inicializovaný,

Modifikátor „in“ v parametri metódy

- Modifikátor **in** – argument sa odovzdáva odkazom
- Je **zaručené**, že sa **argument nezmení**

```
int value = 123;  
Method(value);  
Console.WriteLine(value);
```

123

```
void Method(in int number)  
{  
    // Ak by sme sa pokúsili nastaviť hodnotu parametra,  
    // dostali by sme kompilačnú chybu CS8331  
    //number = 0;  
}
```

Modifikátor „out“ v parametri metódy

- Modifikátor **out** – argument sa odovzdáva odkazom
- Musí byť **inicializovaný** v metóde – **out** zaručuje, že sa hodnota parametra nastaví v jej tele
- Pri volaní metódy sa musí kľúčové slovo **out** uviesť:

```
// Nie je nutné inicializovať premennú,  
// pretože out v metóde zaručí, že sa  
// premenná inicializuje v nej  
int value;  
Method(out value);  
Console.WriteLine(value);  
  
void Method(out int number)  
{  
    number = 0;  
}
```

0

```
var input = Console.ReadLine();  
int number;  
bool success = int.TryParse(input, out number);
```

```
var input = Console.ReadLine();  
// Môžeme deklarovať premennú number aj takto:  
bool success = int.TryParse(input, out int number);
```

```
var input = Console.ReadLine();  
// Môžeme deklarovať premennú number aj takto:  
bool success = int.TryParse(input, out var number);
```

Modifikátor „ref“ v parametri metódy

- Modifikátor **ref** – argument sa odovzdáva odkazom
- Musí byť **inicializovaný** pred volaním metódy a môže byť zmenený v metóde
- Pri volaní metódy sa musí kľúčové slovo **ref** uviesť:

```
int value = 1;  
Method(ref value);  
Console.WriteLine(value);  
  
void Method(ref int number)  
{  
    number = number + 1;  
}
```

2

slido



Ako zavoláme metódu s pomenovanými argumentmi width a height?

ⓘ Start presenting to display the poll results on this slide.

slido



**Koľko argumentov môžeme zadať metóde, ak je definovaná ako:
Metoda(double first, params double[] others)?**

ⓘ Start presenting to display the poll results on this slide.

slido



**Ktoré volanie metódy `int.TryParse` je
CHYBNÉ, ak premenná `input` je deklarovaná
správne typu `string`?**

**Metóda je definovaná nasledovne: `bool
TryParse(string? s, out int result)`**

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Konštruktory (constructors) (1)

- **Syntax pre konštruktor** – názov triedy bez návratového typu
- **Modifikátor prístupu** – určuje, kto ho môže použiť (private, protected, public, ...)
- Nie je nutné vytvárať konštruktor pre triedu – vtedy kompilátor vytvorí bezparametrický konštruktor. Ak si ale aspoň jeden konštruktor vytvoríte, kompilátor bezparametrický konštruktor nevytvorí

```
public class MyClass
{
    public MyClass()
    {
    }

    protected MyClass(string parameter)
    {
    }
}
```

```
public class MyNumber
{
    private int _number;
    public MyNumber(int number) => _number = number;
    //...
}
```

```
MyNumber n = new(42);
```

Konštruktory (2) – primárne (primary)

- Od C# 12 / .NET 8 možnosť definovať **konštruktor v deklarácii triedy** bez nutnosti explicitne definovaného konštruktora

```
public class BankAccount(string accountID, string owner)
{
    public string AccountID { get; } = accountID;
    public string Owner { get; } = owner;

    public override string ToString() => $"Account ID: {AccountID}, Owner: {Owner}";
}
```

Konštruktory (3) – tuple

- Pomocou **syntaxe n-tíc (tuple)** môžeme **inicializovať viaceré vlastnosti** v jednom výraze:

```
public record class Book
{
    public Book(string title, string publisher) =>
        (Title, Publisher) = (title, publisher);

    public string Title { get; }
    public string Publisher { get; }
}
```

Konštruktory (4)

- **Inicializátor konštruktora** (constructor initializer) s **this** – umožňuje volať konštruktor z iného konštruktora

```
class Car
{
    private string _description;
    private uint _nWheels;

    public Car(string description, uint nWheels)
    {
        _description = description;
        _nWheels = nWheels;
    }

    public Car(string description) : this(description, 4)
    {
    }
}
```

Statický konštruktor (static constructor)

- Inicializuje statických členov
- **Nemôže mať žiadne modifikátory prístupov**, používa sa iba kľúčové slovo **static**
- **Statický konštruktor je zavolaný automaticky** pred použitím iného člena triedy alebo pred vytvorením prvej inštancie triedy

```
class MyClass
{
    static MyClass()
    {
        // Inicializačný kód
    }

    //...
}
```


Lokálne funkcie (local functions) (1)

- **Lokálna funkcia** je špeciálny typ metódy, ktorá môže byť definovaná a volaná iba v rámci inej metódy alebo funkcie
- Bez statického modifikátora môže pristupovať k premenným v rámci metódy (closure):

```
public static void Method()  
{  
    int z = 3;  
  
    int result = Add(1, 2);  
    Console.WriteLine("Zavolaná lokálna funkcia s výsledkom: {result}");  
  
    int Add(int x, int y) => x + y + z;  
}
```

Zavolaná lokálna funkcia s výsledkom: 6

Lokálne funkcie (2)

- Je možné ich vyvolať iba v rámci metódy, kde sú deklarované
- Môžu byť implementované ako metódy s využitím zložených zátvoriek ({ }) alebo ako výraz na jeden riadok (=>)
- Od C# 8 sa môže použiť modifikátor **static**, ak nepristupujeme k inštančným členom definovaným v triede alebo k lokálnym premenným v metóde

```
public static void Method()  
{  
    static int Add(int x, int y) => x + y;  
  
    int result = Add(3, 7);  
    Console.WriteLine("Zavolaná lokálna funkcia s výsledkom: {result}");  
}
```

Zavolaná lokálna funkcia s výsledkom: 10

Generické metódy (generic methods)

- **Generická metóda** je metóda deklarovaná s **typovými parametrami**
- Typ parametra (**T**) sa definuje pri **použití** (môžeme vynechať, ak sa vie odvodiť – napr. z typov vstupných argumentov)

```
public static void Swap<T>(ref T x, ref T y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

```
int a = 1;
int b = 2;
```

```
GenericMethods.Swap<int>(ref a, ref b);
GenericMethods.Swap(ref a, ref b);
```

```
string s1 = "a";
string s2 = "b";
```

```
GenericMethods.Swap(ref s1, ref s2);
```

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Rozširujúce metódy (extension methods)

- **Metódy**, ktoré rozširujú iné **typy**
- **Nemajú prístup k privátnym** členom typov
- Kompilátor ich konvertuje na volanie statickej metódy

```
namespace ExtensionsForString;  
  
public static class StringExtensions  
{  
    public static int GetWordCount(this string s) => s.Split().Length;  
}
```

```
using ExtensionsForString;  
  
string fox = "the quick brown fox jumped over the lazy dogs";  
int wordCount = fox.GetWordCount(); // StringExtensions.GetWordCount(fox);  
Console.WriteLine($"Počet slov: {wordCount}");
```

Indexery (indexers) (1)

- **Indexery** umožňujú, aby akákoľvek **trieda bola indexovaná podobne ako pole**, ideálne na vlastné kolekcie
- Podobná **syntax ako pri vlastnostiach**, avšak ako **identifikátor** sa používa **this[]**
 - **Kompilátor automaticky generuje vlastnosť** nazvanú **Item**, avšak nie je z kódu prístupná (ak si ale vytvoríte vlastnosť Item, dostanete na mieste indexera chybu CS0102)
 - Je možné mať aj viac indexerov, ale musia sa odlišovať v počte alebo typov parametroch

```
NávratovýTyp this[Typ index]
{
    // get a set časti
}
```

```
NávratovýTyp this[Typ1 index1, Typ2 index2]
{
    // get a set časti
}
```

Indexery (2) – príklad

- Na vygenerovanie kostry kódu vo Visual Studiu je možné použiť skratku (snippet) **indexer**

```
public class MyList
{
    private List<string> data = new();

    public string this[int index]
    {
        get => data[index];
        set => data[index] = value;
    }

    public int this[string value] => data.IndexOf(value);

    public void Add(string value) => data.Add(value);
}
```

```
var myList = new MyList();
myList.Add("A");
myList.Add("B");
myList.Add("C");

Console.WriteLine(myList[1]);
Console.WriteLine(myList["C"]);

myList[0] = "FX";
Console.WriteLine(myList[0]);

myList["FX"] = 0; // CS0200
```

B
2
FX

Vnorené typy (nested types)

- **Vnorený typ** – typ definovaný vo vnútri triedy, štruktúry alebo rozhrania
 - Vnorený typ triedy môže mať ľubovoľný prístupový modifikátor
 - Vnorený typ štruktúry môže byť iba public, internal alebo private

```
public class Container
{
    public class Nested
    {
        Nested()
        {
        }
    }
}
```

```
Container.Nested nest = new Container.Nested();
```


Operátory (operators) – preťažovanie (1)

- Môžeme definovať vlastné verzie operátorov – preťažený operátor sa definuje ako **public static** metóda s kľúčovým slovom **operator**, ktorá prijíma argumenty a vráti hodnotu
 - operátor musí mať aspoň jeden operand typu danej triedy
 - pri porovnávacích operátoroch (**==**, **!=**, **<**, **>**) sa musí vždy preťažiť aj opačný operátor

```
Public class Person
{
    public string Name { get; }

    public Person(string name)
    {
        Name = name;
    }

    public static bool operator ==(Person a, Person b)
    {
        return a.Name == b.Name;
    }

    public static bool operator !=(Person a, Person b)
    {
        return !(a == b);
    }
}
```

Operátory – konverzie (2)

- Konverzie na požadované typy:
 - **implicitná** – automatická konverzia, nie je potrebné pretypovať
 - **explicitná** – nutné použiť operátor pretypovania

```
class Celsius
{
    public double Degrees { get; }
    public Celsius(double degrees) => Degrees = degrees;
}

class Fahrenheit
{
    public double Degrees { get; }
    public Fahrenheit(double degrees) => Degrees = degrees;
    // Konverzia objektu Fahrenheit na Celsius:
    public static implicit operator Celsius(Fahrenheit f)
    {
        return new Celsius((f.Degrees - 32) * 5 / 9);
    }
}

// Použitie:
Fahrenheit f = new Fahrenheit(100);
Celsius c = f; // Implicitná - automatická konverzia
```

```
// Konverzia objektu Point na int:
public static explicit operator int(Point p)
{
    return p.X + p.Y;
}

// Použitie:
Point p = new Point(3, 4);
int sum = (int)p; // Explicitná konverzia
```

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Gajdlajny pre programovanie v C#

- **Identifikátory**

- Začínajú písmenom alebo podčiarkovníkom
- Nemôžu používať kľúčové slová C#

- **Konvencie pomenovania**

- „**Pascal casing**“ – prvé veľké písmená
 - **menné priestory** (napr. Microsoft.EntityFrameworkCore, ...)
 - **typy** (napr. DateTime, SqlCommand, ...)
 - **vlastnosti** (napr. Length, FirstName, IsLoggedIn, DateOfBirth, ...)
 - **metódy** (napr. ToUpper(), IndexOf(), IsNullOrEmpty(), ...)
 - **konštanty** (napr. Int32.MaxValue, Math.PI, Math.Tau, ...)
- „**Camel casing**“ – prvé malé písmeno
 - **dátové členy** typov (atribúty z OOP) (napr. _firstName, firstName, ...)
 - **parametre** metód (napr. args, firstName, dateOfBirth, ...)
 - lokálne **premenné** (napr. i, j, length, count, ...)



Použitá literatúra

- Christian Nagel: **Professional C# and .NET** 2021 Edition, 8. vydanie, ISBN-13: 978-1119797203
- .NET a C#:
 - <https://learn.microsoft.com/en-us/dotnet/>
 - <https://learn.microsoft.com/en-us/dotnet/csharp/>



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Katedra softvérových
technológií

stefan.toth@uniza.sk

ĎAKUJEM ZA POZORNOSŤ

Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu **Jazyk C# a .NET** na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Ing. **Štefan Toth**, PhD.
stefan.toth@uniza.sk