C# - Úvod, datové typy, základní konstrukce

Ing. Roman Diviš

UPCE/FEI/KST

Obsah

1. Použitý styl v přednáškách

2. Datové typy, výrazy, základní konstrukce
   - Proměnné, datové typy
   - Výrazy
     - Základní řídící konstrukce
   - Pole
   - Základní konzolové příkazy

Použitý styl v přednáškách

Základní informace

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Příklad

Nullam mattis efficitur aliquam.

Chyba / příklad s chybou / nekorektní použití

Sed aliquam iaculis massa, vel tincidunt lacus tincidunt eget.

Poznámka (teorie, vhodné k zapamatování)

Proin porta urna ut ipsum ornare, a ultricies elit dictum.

Deprecated (staré, už nepoužívané)

Pellentesque habitant morbi tristique senectus et netus et malesuada fames.

Bonus (nepotřebujete ke zkoušce, ale souvisí s tématem)

Sed imperdiet pharetra est, sed ullamcorper neque.

Ukázka

Ukazatele a dynamická paměť

Ukázky korektní a nekorektní práce s ukazateli a dynamicky alokovanou pamětí:

✓ Good

```
int* pointer = nullptr;
pointer = new int;
*pointer = 123;
```

× Bad

```
int* pointer = 0xdeadbeef;
*pointer = 123;
```

⚠ deprecated

```
int* pointer = (int*)malloc(sizeof(int));
*pointer = 123;
```

Definice kódu

```
struct|class nazevDatovehoTypu [final] [dědičnost] {
  [složky – atributy, metody, vnořené typy]...
} [objekty];

[dědičnost]:
  : [viditelnost] [virtual] předek1, ...
```

Definuje:

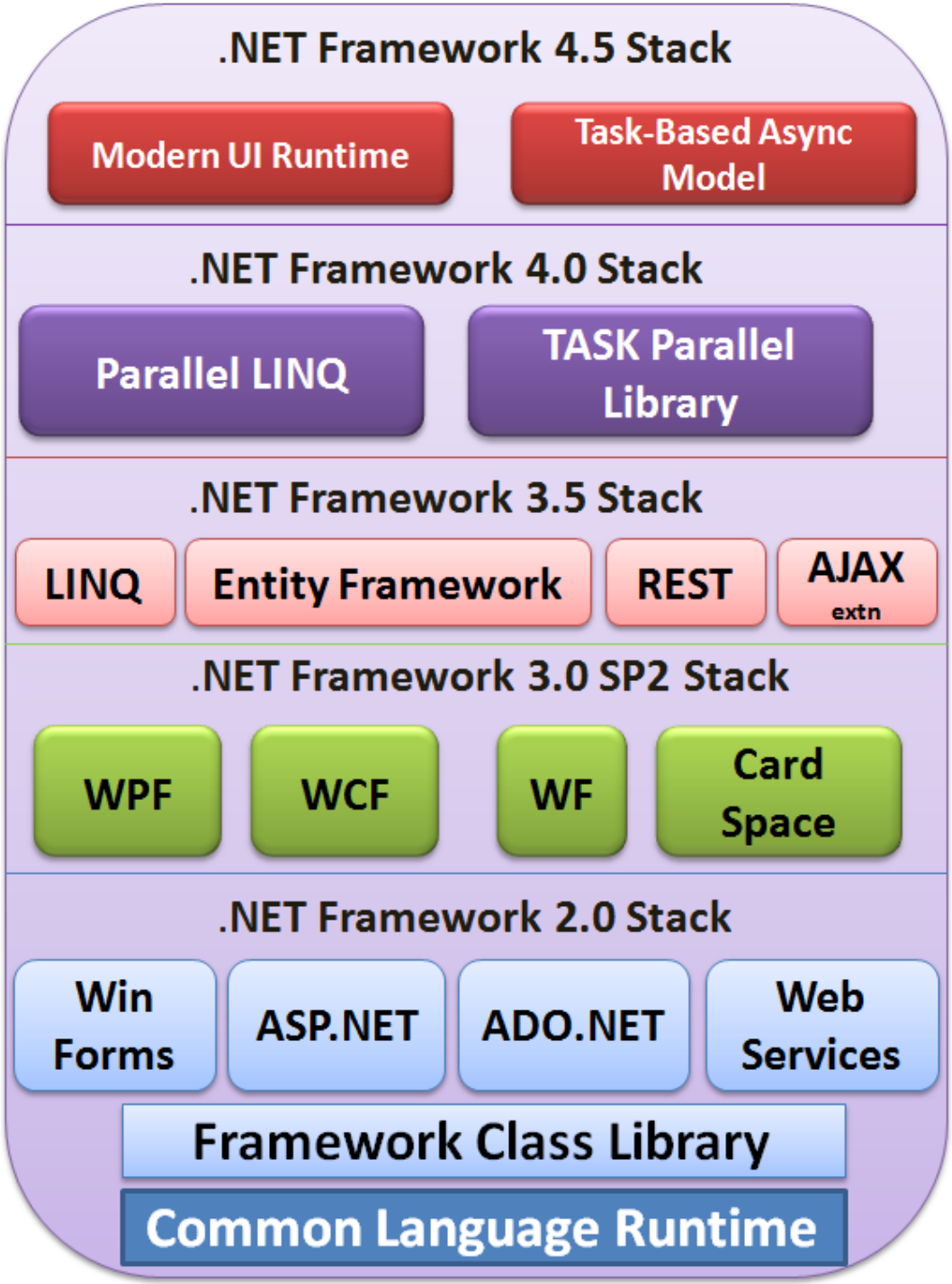
- Začínáme klíčovým slovem **struct** nebo **class**
- dále uvedeme název datového typu (`Pes`, `Kocka`, `Student`, ...)
- dále může být (ale nemusí) klíčové slovo `final`
- dále může být definováno dědění z předků

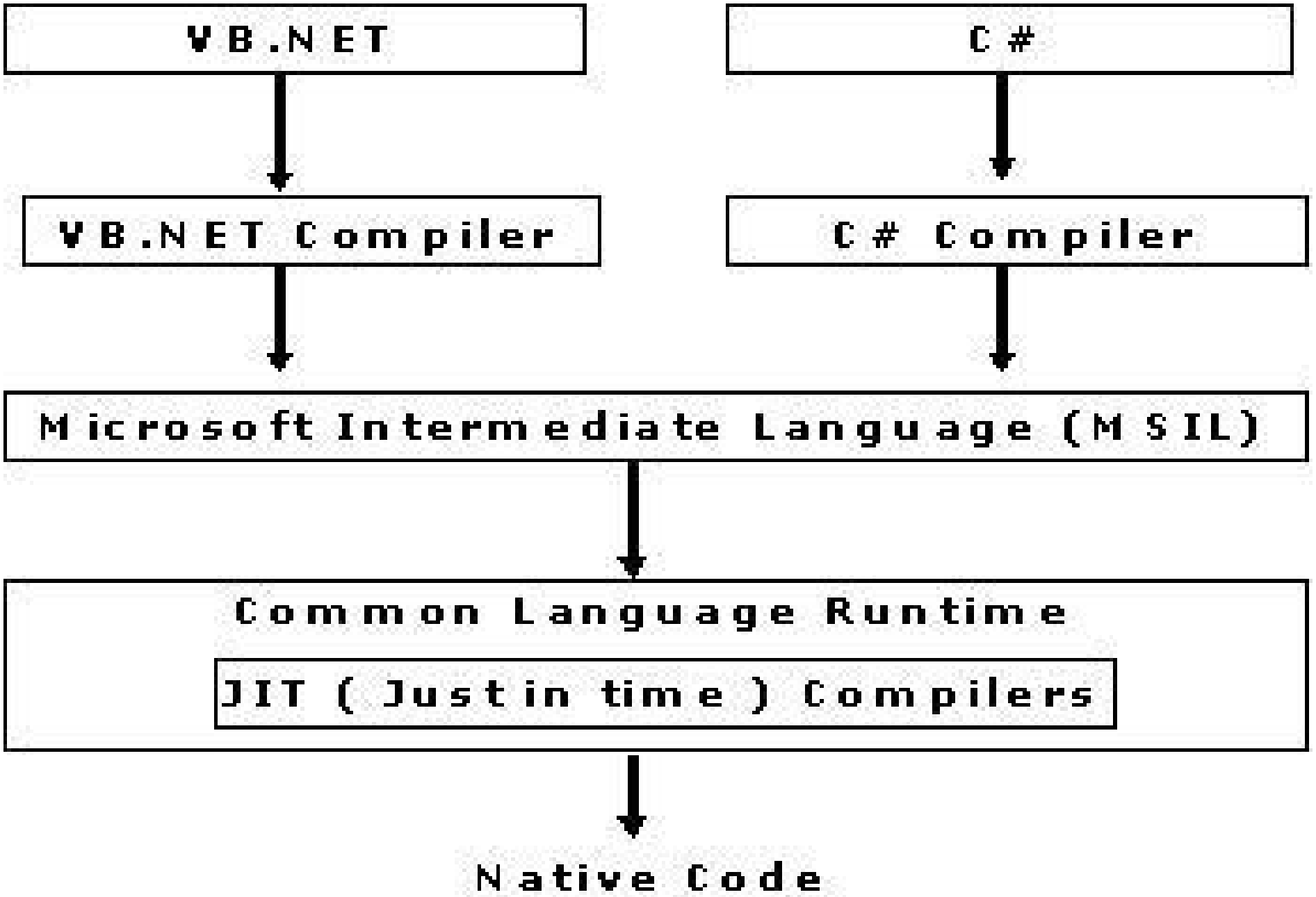
- Uvnitř třídy je možné definovat větší množství složek (...)

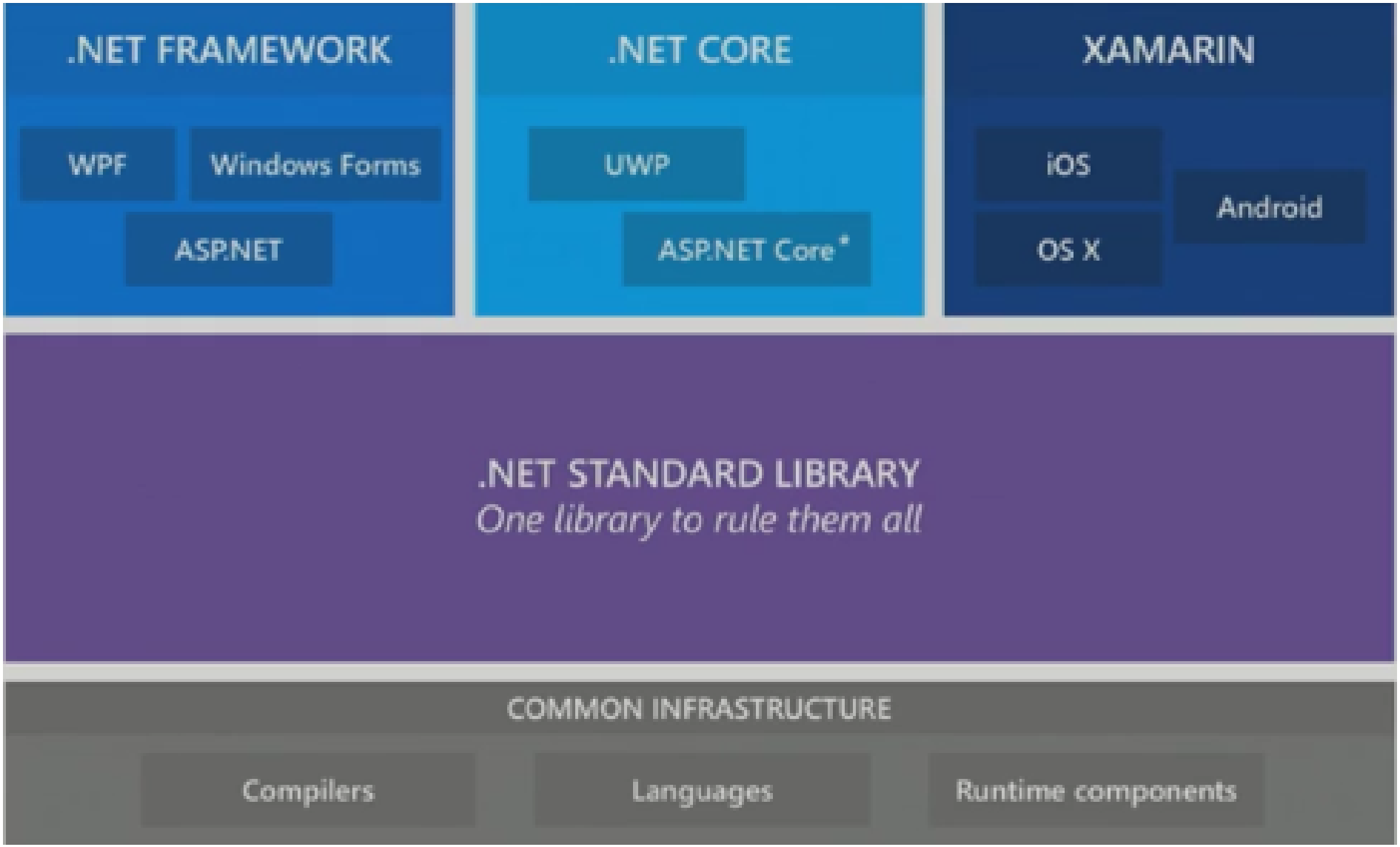
Slidy označené fialovými pruhy obsahují téma, které není vyžadováno u zkoušky a zápočtu.

Datové typy, výrazy, základní konstrukce

Java	C/C++	C#
	Výstup	
bytecode	nativní kód	MSIL (bytecode)
	Správa paměti	
garbage collector reference	ručně ukazatele, reference	garbage collector ukazatele, reference
	Objektový model	
jednoduchá dědičnost	vícenásobná dědičnost	jednoduchá dědičnost vlastnosti, indexery, delegáty, události
	Přetěžování operátorů	
×	✓	✓
	Organizace modulů	
package Maven, Gradle	namespace ...	assembly,namespace NuGet
	Generické programování	
genericita (type erasure)	šablony	genericita (reification)
	Knihovna jazyka	
java.lang, java.util, ...	C/C++, STL	java like


.NET Framework 4.5 Stack
Modern UI Runtime
Task-Based Async Model
.NET Framework 4.0 Stack
Parallel LINQ
TASK Parallel Library
.NET Framework 3.5 Stack
LINQ
Entity Framework
REST
AJAX extn
.NET Framework 3.0 SP2 Stack
WPF
WCF
WF
Card Space
.NET Framework 2.0 Stack
Win Forms
ASP.NET
ADO.NET
Web Services
Framework Class Library
Common Language Runtime


VB.NET
C#
VB.NET Compiler
C# Compiler
Microsoft Intermediate Language (MSIL)
Common Language Runtime
JIT (Just in time) Compilers
Native Code


.NET FRAMEWORK
WPF
Windows Forms
ASP.NET
.NET CORE
UWP
ASP.NET Core*
XAMARIN
iOS
Android
OS X
.NET STANDARD LIBRARY
One library to rule them all
COMMON INFRASTRUCTURE
Compilers
Languages
Runtime components

C#

- kompilovaný jazyk (MSIL - mezikód)
  - do nativního kódu je převeden runtime pomocí JIT
- statické (i dynamické) typování
- správa dynamické paměti pomocí garbage collectoru
- základní knihovna podobná Javovské
- inspirován Javou, C/C++, ...
- aktuální verze C# 7.2
- standard a některé komponenty uvolněny pod open source licencemi
- podpora pro non-Windows platformy pomocí Mono, nově .NET Core

Hello world!

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Hello world! a dokumentační komentáře

```
namespace ConsoleApplication1
{
    /// <summary>
    /// Hlavni program.
    /// </summary>
    class Program
    {
        /// <summary>
        /// Main metoda
        /// </summary>
        /// <param name="args">argumenty prikazove radky</param>
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Coding conventions

- C# definuje řadu doporučení a pravidel, jakým stylem by měl být psán a formátován kód
- Základní doporučení:
  - VeřejnéTypy, Třídy, Struktury, Metody, Vlastnosti, ... začínají velkým písmenem, používá se PascalCase
  - privátníAtributy, parametryMetod, lokálníProměnné, ... začínají malým písmenem, používá se camelCase
  - nepoužívá se hungarian notation, prefix `m_`, ...

✓

```
class CarFactory { }
struct StudentRecord { }

public void GetElementAt(int elementIndex) { }

private string name;
```

Proměnné, datové typy

Primitivní datové typy, proměnné

- existují hodnotové a referenční datové typy
  - hodnotové typy jsou při předávání do metod nebo ve výrazech typu "`promenna = puvodniHodnota`"předány/vytvořeny jako kopie původní hodnoty – vzniká nová instance (objekt) daného typu se stejnou hodnotou (stavem)
  - referenční typy jsou předány jako reference, tj. existuje jediná instance daného objektu a manipulace s jejich stavem z nového i původního umístění mění jeden a ten samý objekt
- lze používat reference (na hodnotové typy, **ref**)
- lze používat ukazatele (**unsafe**)
- lokální proměnné se pojmenovávají dle camelCase (pocetZivotu, maximalniVykonMotoru)

```
nazevDatovehoTypu nazevPromenne [= inicializacniHodnota];
```

Primitivní datové typy

```
// celá čísla
int x = 123;
// znakový typ
char c = 'c';
// desetinné typy
double d = 3.141592;
float f = 3.141592f;
// řetězec (není primitivní typ)
string s = "hello world";
// logický typ
bool b = true;
// desitkový desetinný typ
decimal dec = 3.333M;

Console.WriteLine("{0} {1} {2} {3} {4} {5} {6} {7} {8}", x, c, d, f, dec↩
    , va, nullablex, s, b);
Console.WriteLine($"{x}, {c}, {d}, {f}, {s}, {b}, {dec}");
```

Primitivní datové typy

```
// uint je 32 bitový bezznaménkový typ
uint bezznamenkovyInteger;
```

```
System.SByte int8; // sbyte
System.Int16 int16; // short
System.Int32 int32; // int
System.Int64 int64; // long

System.Byte uint8; // byte
System.UInt16 uint16; // ushort
System.UInt32 uint32; // uint
System.UInt64 uint64; // ulong

System.Single fSingle; // float
System.Double fDouble; // double
System.Decimal fDecimal; // decimal
```

Synonyma

- Pozor na synonyma, je možné používat obojí, ale preferována je použití klíčového slova před názvem typu...
- **string** == `String`
- **object** == `Object`
- **int** == `System.Int32`
- Tedy používejte **string**, **object**, **int**

Nulovatelné datové typy

- `datovyTyp?`
- slouží jako „rozšiřující modifikátor“ pro uložení hodnoty **null** do libovolného typu
- objekt pak má vlastnosti `HasValue`, `Value` a metodu `GetValueOrDefault`

✓

```
// int může nabývat hodnot −2147483648 až 2147483647
// co když chceme říct, že hodnota není?
int? nulovatelnyInt = null;

nulovatelnyInt = 100;
nulovatelnyInt = null;
```

var – automatické odvození datového typu

- **var** umožňuje nechat kompilátor odvodit datový typ
- nejedná se o dynamický typ, **var** nabývá pouze konkrétního datového typu
- umožňuje použití anonymních (objektových) datových typů

✓

```
var varPromenna = 123;
// varPromenna je int
varPromenna++;
// nelze: varPromenna = "abcd";
```

Výčtové typy

✓

```
enum VyctovyTyp
{
    Prvni, // = 0
    Druhy, // = 1
    Treti // = 2
}

Console.WriteLine($"{VyctovyTyp.Prvni}");
```

✓

```
enum VyctovyTyp
{
    Prvni = 10,
    Druhy, // = 11
    Treti // = 12
}
```

Výčtové typy – bitové masky

```
[Flags] // atribut System.FlagsAttribute
enum Permissions
{
    Read = 0x01,
    Write = 0x02,
    Execute = 0x04
}

static void Main(string[] args)
{
    Permissions p = Permissions.Read | Permissions.Write;
    if (p.HasFlag(Permissions.Read))
    {
        Console.WriteLine($"{p}");
    }
}
```

Výrazy

Proměnné a výrazy/operátory

```
int x = 123;

// přiřazení, složené výrazy
// =, +, -, *, /, %, +=, -=, *=, /=, %=
x = 456;
x += 99;
x = x - 99;
x = x * x;
// inkrementace/dekrementace
x++;
--x;
// ternární operátor
x = (x % 2 == 0) ? x : 0;

Console.WriteLine($"{x}");
```

Proměnné a výrazy/operátory

```
int x = 123;
int y = 456;

// porovnání
// >, <, >=, <=, ==, !=
bool vysledekVetsi = x > y;
bool vysledekShoda = x == y;

// bitové operátory
// &, |, ^, >>, <<
int z = x & (y << 5);
```

Konverze datového typu – kulaté závorky

```
double dbl = 123.45;
int ine = (int)dbl;
```

```
Parent parent = new Child();
Child child;

// vyvolá InvalidCastException pokud objekt nebude typu Child
child = (Child)parent;
```

Konverze datového typu – as, is

```
Parent parent = new Child();
Child child;

// null pokud nebude daného typu
child = parent as Child;

// is – C# obdoba operátoru instanceof z Javy
if (parent is Child)
{
    Child cld = (Child)parent;
    cld.Action();
}

// C# 7 – umožňuje rovnou deklarovat proměnnou
if (parent is Child chd)
{
    chd.Action();
}
```

Konverze datového typu – reflexe + typeof

```
Parent parent = new Child();
Child child;

if (parent.GetType() == typeof(Child))
{
    child = (Child)parent;
}
```

Null operátory

```
int? a = null;

// ??
int b = a ?? -1;
// a == null? potom b = -1
// a != null? potom b = a

// ?.
object o = null;
o?.ToString();
// o == null? potom nic nedělej
// o != null? potom zavolej ToString()

// ?[]
int[] array = null;
int? x = array?[0];
```

✓

```
var value = firstObject?.secondObject?.maybeArray?[0] ?? "default";
```

Přetečení/podtečení – checked blok

```
int i = 10, j = 0;
i = i / j; // vyvolá DivideByZeroException

(i, j) = (int.MaxValue, int.MinValue); // Tuples/n-tice C# 7
i++;
j--;
Console.WriteLine($"{i} {j}");
// i == -2147483648 == int.MinValue
// j == 2147483647 == int.MaxValue

(i, j) = (int.MaxValue, int.MinValue);
checked
{
    i++; // vyvolá OverflowException
    j--; // vyvolá OverflowException
}
```

Základní řídící konstrukce

Rozhodování

- syntax stejná jako Java/C/C++
- podmínka musí být vyhodnocena na typ **bool**

✓

```
int x = 123;
if (x > 10)
{
    Console.WriteLine("x > 10");
}
```

×

```
int x = 123;
if (x)
{
    Console.WriteLine("error");
}
```

✓

```
int x = 123;
if (x > 10)
{
    Console.WriteLine("x > 10");
}
else
{
    Console.WriteLine("x <= 10");
}
```

Cykly

- **while**, **do–while**, **for**, **foreach**
- syntax shodná
- **foreach** lze využívat na pole a kolekce
- podporováno řízení cyklů **break**, **continue**

✓

```
int x = 123;
for (int i = 0; i < x; i++)
{
    Console.WriteLine("{0}", i);
}
```

✓

```
do
{
    x--;
} while (x > 0);
```

✓

```
while (x < 10)
{
    x++;
}
```

✓

```
int x = 123;
while (true)
{
    if (x == 200)
        continue;

    if (x == 300)
        break;

    x++;
}
```

Switch

- syntax shodná
- nelze propadávat z jednoho **case** do druhého, je nutné explicitně skočit **goto case** 123
- nepovinná větev **default** pro ostatních hodnot

- od C# 7 podpora pattern matching
  - umožňuje rozlišit datový typ objektu (lze dále definovat podmínku)
  - **case string** `s` **when** `s.Contains`("foobar"):
  - **case int** `n` **when** `n` $>$ 100:
  - **case int** `n`:

✓

```
int x = 123;
switch (x)
{
    case 0:
        Console.WriteLine("0");
        break;

    case 1:
        Console.WriteLine("1");
        //goto case 0; // preskok na 0
        //goto default; // preskok na default
        break;

    default:
        Console.WriteLine("...");
        break;
}
```

Pole

Pole

- pole je **referenční objekt**
  - při předání do metody je možné měnit obsah pole (stejně jako v Javě)
- C# umožňuje vytvářet 3 typy polí
  - jednorozměrná pole
  - vícerozměrná pole (pravidelná)
  - vícerozměrná pole (nepravidelná, jagged array)
- syntax obdobný Javě, hranaté závorky, indexování prvků od nuly, přístup mimo rozsah pole vyvolá výjimku

Jednorozměrné pole

✓

```
int[] pole;
pole = new int[10];

pole[0] = 10;
pole[1] = 20;
//pole[10] = 15; // IndexOutOfRangeExpection

int pocetPrvkuPole = pole.Length;

Console.WriteLine($"{pocetPrvkuPole} {pole.Length} {pole[0]}{pole[1]}");
```

Vícerozměrné pole (pravidelné)

✓

```
int[,] pole = new int[2,3];

pole[0, 0] = 1;
pole[0, 1] = 2;
pole[0, 2] = 3;
pole[1, 0] = 4;
pole[1, 1] = 5;
pole[1, 2] = 6;

int pocetPrvkuVPoli = pole.Length; // == 2*3 == 6
int pocetPrvkuVPrvniDimenzi = pole.GetLength(0); // == 2
int pocetPrvkuVDruheDimenzi = pole.GetLength(1); // == 3

Console.WriteLine($"{pocetPrvkuVPoli} {pole.Length} {↩
    pocetPrvkuVPrvniDimenzi} {pocetPrvkuVDruheDimenzi} {pole[0, 0]} {↩
    pole[0, 1]}");
```

Vícerozměrné pole (pravidelné)

✓

```
int[] pole1D = new int[2];
int[,] pole2D = new int[2,3];
int[,,] pole3D = new int[2,3,5];
int[,,,] pole4D = new int[2,3,5,6];
int[,,,,] pole5D = new int[2,3,5,6,7];
```

Vícerozměrné pole (nepravidelné, jagged array)

✓

```
int[][] jaggedArray = new int[3][];
jaggedArray[0] = new int[2];
jaggedArray[1] = new int[3];
jaggedArray[2] = new int[4];

jaggedArray[0][0] = 1;
jaggedArray[0][1] = 2;
jaggedArray[1][0] = 3;
jaggedArray[1][1] = 4;
jaggedArray[1][2] = 5;

Console.WriteLine($"{jaggedArray[0][0]} {jaggedArray[0][1]}");
```

Základní konzolové příkazy

Výstupní parametry metod

Umožňují předat volající funkci další výsledky.

✓

```
static void OutMethod(int a, out int b, out int c)
{
    b = 2 * a;
    c = 3 * a;
}

static void Main(string[] args)
{
    int a = 123;
    int b, c;
    OutMethod(a, out b, out c);
    Console.WriteLine("{0} {1} {2}", a, b, c);
}
```

Referenční parametry metod

Parametry funkce předávané "odkazem".

✓

```
static void RefMethod(int a, ref int b)
{
    b = b * a;
}

static void Main(string[] args)
{
    int a = 123;
    int b = 456;
    RefMethod(a, ref b);
    Console.WriteLine("{0} {1}", a, b);
}
```

Konverze string → int/double/...

```
string str = "123";
// Parse – vyvolá FormatException, pokud parametr neobsahuje řetězec s čí↩
    slem
int istr = int.Parse(str);
```

Konverze string → int/double/...

```
// TryParse – vrací bool (úspěch/neúspěch konverze), zkonvertovaná hodnota↩
    je vrácena výstupním parametrem
int istr2;
bool uspech = int.TryParse(str, out istr2);
if (uspech)
{
    Console.WriteLine(istr2);
}
```

Také lze využít třídu System.Convert

```
int valueInt = Convert.ToInt32("123456");
double valueDouble = Convert.ToDouble("3,1415");
```

Konverze int/double/... → string

```
int i = 123;

string s = i.ToString();
// lze také:
// string s = 123.ToString();
```

Třída System.Console

```
Console.Write(...); // vytiskne text
Console.WriteLine(...); // vytiskne text a odradkuje
Console.Read(); // nacte znak
Console.ReadKey(); // nacte stisk klavesy
Console.ReadLine(); // nacte radek textu
```