JAZYK C# – ÚVOD

Literatura

1. VIRIUS, M. *Od C++ k C#*. České Budějovice: KOPP, 2002. 235 s. Cena 199 Kč. ISBN 80-7232-176-5.
2. NAGEL CH. et al. *C# 2005. Programujeme profesionálně*. Brno: Computer Press, 2007. 1400 s. Cena 1390 Kč. ISBN 80-251-1181-4.
3. TROELSEN, A. *C# a .NET 2.0 profesionálně*. Brno: Zoner Press, 2007. 1376 s. Cena 950 Kč. ISBN 80-86815-38-2.
4. ECMA-334. *The C# Language Specification* [online]. 4th edition, June 2006. URL: <http://www.ecma-international.org/publications/standards/Ecma-334.htm> [citováno 2007-02-19].

Jazyk C#

- Čistě objektový jazyk.
- Navržený firmou Microsoft.
- Zveřejněný v roce 2000.
- Programy určené pro platformu .NET.

.NET Framework

- Nadstavba operačního systému.
- Zkvalitňuje služby OS a zvyšuje jeho bezpečnost.
- Základem je **společný běhový systém (Common Language Runtime – CLR)**. Umožňuje běh a vzájemnou spolupráci aplikaci napsaných v různých programovacích jazycích.
- Součásti CLR:
  - **Společná specifikace jazyka (Common Language Specification – CLS)** – pravidla pro tvorbu překladačů jakéhokoli jazyka určeného pro prostředí .NET Framework.
  - **Společný typový systém (Common Type System – CTS)** – pravidla pro definici datových typů a zacházení s nimi.
  - **Automatická správa paměti (Garbage Collector)** – provádí dealokaci objektů, na které již neexistuje žádný odkaz. Dealokace se nemusí provést ihned po zrušení odkazu na objekt.
- Programy se nepřekládají do strojového kódu, ale do **mezijazyka (Intermediate Language – IL,** někdy **Common IL – CIL)**. Obdoba bajtového kódu jazyka Java. Jazyk IL se ale neinterpretuje, ale na cílovém počítači se překládá do strojového kódu pomocí **překladačů JIT (Just In Time)**. Překladač JIT kompiluje každou z částí programu až v okamžiku jejího volání a výsledný zdrojový kód uloží. Jednou přeložená část programu se při příštím využití znovu do strojového kódu nepřekládá. Překladač JIT provádí optimalizovanou kompilaci využívající plně funkcí konkrétního cílového procesoru.
- Nad CLR stojí **základní knihovna tříd (Basic Class Library – BCL)** a specializované knihovny.
- Nejvyšší vrstvu tvoří překladače programovacích jazyků.

- Programuje se v **řízeném kódu (managed code)**. Programování bez použití .NET je v nativním kódu (native code). Řízený kód obsahuje kromě instrukcí IL metadata – informace o datových typech v kódu deklarovaných.
- Jednotka programu vytvořená překladačem (exe, dll) se nazývá **sestavení (assembly)**. Obsahuje řízený kód a **manifest** sestavení – metadata, obsahující informace o aktuální verzi sestavení, informace o využívaných sestaveních, kultuře (jazykové mutaci) aj.
- Druhy sestavení:
  - soukromé – určené pro jednu aplikaci – jsou umístěny ve složce aplikace
  - sdílené – určené pro více aplikací – jsou umístěny v **globálním zásobníku sestavení (Global Assembly Cache – GAC)**. V GAC může existovat více stejných sestavení lišících se svou verzí. GAC bývá v adresáři c:\windows\assembly
- Program ILDASM (Intermediate Langauge Disassembler) – slouží pro prohlížení obsahu sestavení. Lze jej spustit pomocí nabídky **Start | Microsoft .NET Framework SDK v2.0 | Tools | MSIL Disassembler**.
- Proces aplikace může být tvořen několika **aplikačními doménami**. Aplikační domény jednoho procesu sdílejí společný virtuální adresový prostor. Předávání dat mezi aplikačními doménami jednoho procesu je výkonnější než předávání dat mezi procesy. Pokud jedna aplikační doména způsobí havárii, ostatní aplikační domény stejného procesu mohou běžet dál. Aplikační domény lze použít např. u serveru, který pro každého klienta má svou vlastní aplikační doménu.
- Existují i platformy .NET pro jiné operační systémy:
  - Projekt Mono – www.mono-project.com
  - Projekt Portable.NET – www.dotgnu.org.

  Projekty využívají mj. normu ECMA-335 – **Common Language Infrastructure (CLI) – společná infrastruktura jazyků**, která popisuje prostředí .NET Framework.

První program

Následující program je konzolová aplikace, která vypíše na obrazovku dva řádky textu. Program se ukončí stisknutím libovolné klávesy.

Příklad

```
using System;

namespace PrvniProgram
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Toto je můj první program.");
         Console.WriteLine("Konec programu - libovolná klávesa");
         Console.ReadKey();
      }
   }
}
```

Komentář

Druhy komentářů:

- jednořádkový – začíná znaky //

- víceřádkový – začíná znaky `/*` a končí znaky `*/` – nelze do sebe vnořovat.
- dokumentační – začíná znaky `///` a končí na konci řádku.

Dokumentační komentáře

Dokumentační komentář obsahuje předdefinované značky jazyka XML, mezi něž patří zejména následující:

- `<summary>` – souhrnné informace o typu nebo jeho složce.
- `<returns>` – popis návratové hodnoty metody.
- `<param>` – popis parametru metody (syntaxi ověřuje překladač).
- `<paramref>` – odkaz na název parametru metody.
- `<remarks>` – text poznámky.
- `<value>` – popis hodnoty vlastnosti třídy (struktury).
- `<see>` – odkaz na jiný identifikátor (metodu, třídu apod.) (syntaxi ověřuje překladač).
- `<seealso>` – označuje část „Viz také“, obsahující odkaz na jiný identifikátor (syntaxi ověřuje překladač).
- `<c>` – text označený jako kód (klíčové slovo).
- `<code>` – více řádků označených jako kód.
- `<example>` – označuje sekci příkladu kódu.
- `<exception>` – popisuje typ výjimky a stav, kdy může vzniknout.
- `<para>` – označuje nový odstavec.
- `<list>` – označuje seznam odrážek.

Dokumentační komentář k typu, jeho složce a parametru metody je zobrazován ve vývojovém prostředí při použití daného objektu.

Generování XML souboru dokumentačních komentářů – menu **Project | Properties | Build,** zaškrtávací políčko **XML documentation file**.

Převod XML souboru dokumentačních komentářů do souboru nápovědy – program **Microsoft Sandcastle** – konzolová aplikace, volně dostupná – složitá obsluha. Pro něj existují různé GUI nadstavby od jiných firem – viz http://www.sandcastledocs.com.

Příklad

```
/// <summary>
/// Třída sloužící jako příklad použití dokumentačních komentářů.
/// </summary>
/// <remarks>
/// Třída obsahuje dvě statické metody:
/// <see cref="DruhaMocnina"/> a <see cref="Main"/>.
/// </remarks>
class Program
{
   /// <summary>
   /// Metoda pro výpočet druhé mocniny čísla typu <c>int</c>.
   /// </summary>
   /// <param name="x">Umocňované číslo.</param>
   /// <returns>Druhá mocnina parametru <paramref name="x"/>.</returns>
   static int DruhaMocnina(int x)
   {
      return x * x;
   }
   /// <summary>
   /// Metoda, kterou se začíná běh programu.
   /// </summary>
   /// <exception cref="System.IO.IOException">
   /// Chyba při výpisu na obrazovku.</exception>
   static void Main()
   {
      Console.Write("Druhá mocnina 5 je " + DruhaMocnina(5));
   }
}
```

Konvence pojmenování

- Pascalovský styl (angl. pascal casing) – první písmeno každého slova velkým písmenem bez podtržítek, např. `DruhaMocnina` – pro většinu identifikátorů.
- Velbloudí styl (angl. camel casing) – první písmeno malé, druhé a další slovo začíná velkým písmenem, např. `druhaMocnina`. Použít pro následující identifikátory:
  - parametry metod,
  - lokální proměnné,
  - soukromé datové složky třídy.
- Prostory jmen:
  - hlavní prostor jmen aplikace – název společnosti
  - vnořený prostor jmen – název technologie (aplikace).
- Názvy identifikátorů bez prefixu, např. název třídy by neměl být `CTrida` nebo `TTrida`, ale jen `Trida`. Výjimku tvoří interface, který má prefix `I`, např. `IDisposable`.
- Zkratky a akronymy mají být psány malými písmeny, kromě prvního znaku, který závisí na typu identifikátoru (viz výše). Např. `htmlControl` pro soukromé složky resp. `HtmlControl` pro ostatní složky. Výjimkou jsou dvouznaké zkratky, které by měly být psány velkými písmeny, kromě identifikátorů, které začínají zkratkou a mají být ve velbloudím stylu. Např. veřejná složka `DBCommand`, soukromá složka `dbCommand`. Ale v knihovně .NET není toto pravidlo dodrženo, např. třída `DbConnection`. Pravidlo je dodrženo u prostorů jmen a typů, např. prostor jmen `System.IO`, třída `GC`.

- Název výčtového typu:
  - obsahuje příznaky – v množném čísle, např. `MouseButtons`
  - neobsahuje příznaky – v jednotném čísle, např. `DialogResult`.
- Další informace – viz knihovna MSDN, téma „Guidelines for Names“ (naming guidelines) a další témata přednášek.

Identifikátory

- Posloupnost písmen a číslic, začínající písmenem.
- Za písmeno se považuje jakýkoliv znak v kódování UNICODE.
- Rozlišují se malá a velká písmena.
- Před identifikátor lze připojit znak `@`, který říká, že se jedná o identifikátor. Např. `@Číslo` a `Číslo` jsou stejné identifikátory. Znak `@` lze použít, pokud jako identifikátor chceme použít klíčové slovo, např. `@do` je identifikátor, `do` je klíčové slovo.

Klíčová slova

Všechna klíčová slova jsou malými písmeny. Seznam viz knihovna MSDN.

Deklarace proměnné

Neexistují globální proměnné.

Syntaxe:

deklarace_jedné_proměnné:

modifikátory$_{nep}$ *typ identifikátor inicializace*$_{nep}$**;**

Modifikátory – `const` nebo `volatile`.

Typ – typ proměnné.

Identifikátor – jméno proměnné.

Inicializace – nepovinná inicializační část proměnné – dtto C++.

Příklady:

```
int i, j = 11, k = 5;
const int m = 10, n = 20; // n je také konstanta
```

Rozdíly oproti C++:

- Modifikátor `const` lze použít pro předdefinované hodnotové typy (základní datové typy), výčtové typy a typ `string`. Pro referenční typy (třídy apod.) lze modifikátor `const` použít pouze k deklaraci proměnné, která je inicializována hodnotou `null`.
- Překladač nedovolí použít proměnnou, která nebyla inicializována.
- Ve vnořeném bloku nelze deklarovat proměnnou stejného jména jako v nadřízeném bloku.

Prostory jmen

Rozdíly oproti C++:

- Mezi prostorem jmen a identifikátorem a mezi vnořeným a nadřízeným prostorem jmen se uvádí tečka.
- Pro složky prostorů jmen lze specifikovat přístupová práva.

Syntaxe:

deklarace prostoru jmen:

namespace *jméno* **{** *direktiva_using*$_{nep}$ *deklarace_složek* **}**

Jméno – jméno prostoru jmen.

Direktiva_using – direktiva `using` – viz dále.

Deklarace_složek – deklarace datových typů (tříd, struktur, výčtových typů aj.) a vnořených prostorů jmen.

Vnořený prostor jmen lze deklarovat dvěma způsoby.

První způsob:

```
namespace Vnější
{
   namespace Vnitřní
   {
      class X {}
   }
}
```

Druhý způsob (neexistuje v C++):

```
namespace Vnější.Vnitřní
{
   class X {}
}
```

Na identifikátor `X` se mimo prostor jmen odvoláváme zápisem `Vnější.Vnitřní.X` v obou uvedených případech deklarace.

Identifikátory, které nejsou deklarovány v žádném prostoru jmen, jsou součástí *globálního prostoru jmen*.

Přístupová práva

Před deklarací datového typu v prostoru jmen může předcházet specifikace přístupových práv:

- `public` – datový typ je veřejně přístupný, tj. lze ho použít i v jiných sestaveních,
- `internal` – datový typ lze použít pouze v sestavení, v němž je deklarován.

Neuvede-li se žádná specifikace, překladač doplní specifikaci `internal`.

Direktiva using

Existují tři verze deklarace direktivy `using`.

Syntaxe:

direktiva_using:

using *jméno_prostoru_jmen***;**

using *alias* **=** *jméno_prostoru_jmen***;**

using *alias* **=** *jméno_typu***;**

Direktiva `using` musí předcházet všem deklaracím v prostoru jmen. Totéž platí i pro globální prostor jmen.

První verze

První verze direktivy slouží k specifikaci prostoru jmen, jehož identifikátory se nemusí v programu kvalifikovat jménem tohoto prostoru.

Je-li v programu uvedena direktiva

```
using System;
```

lze psát

```
Console.WriteLine("Nějaký text ");
```

protože třída `Console` je součástí prostoru jmen `System`.

Nelze ale napsat

```
File.Create("data.txt");
```

ani

```
IO.File.Create("data.txt");
```

protože třída `File` leží v prostoru jmen `System.IO` a ten není direktivou `using` zpřístupněn.

Druhá verze

Druhá verze slouží k jinému označení prostoru jmen.

Alias – nové jméno prostoru jmen.

Jméno_prostoru_jmen – původní jméno prostoru jmen.

Např. pokud se použije tato deklarace `using`

```
using VV = Vnější.Vnitřní;
```

lze potom přistoupit ke složce `X` výrazem `VV.X`.

Třetí verze

Třetí verze slouží k deklaraci nového jména pro daný typ. V jazyce C/C++ se používá deklarace `typedef`.

Alias – nové jméno typu.

Jméno_typu – původní jméno typu.

Např. lze napsat

```
using VVX = Vnější.Vnitřní.X;
```

Nelze však přejmenovat klíčové slovo, např.

```
using cela = int;
```

Lze ale napsat

```
using cela = System.Int32;
```

Typ `int` je jiné označení pro typ `Int32` v prostoru jmen `System`.

Operátor ::

Operátor `::` (angl. *namespace alias qualifier*) byl zaveden od verze .NET 2.0. Lze jej použít ve spojení s klíčovým slovem `global` nebo ve spojení s aliasem prostoru jmen deklarovaného pomocí direktivy `using`.

Klíčovým slovem `global` lze kvalifikovat složku, která je součástí globálního prostoru jmen. Využívá se v případě, kdy složka globálního prostoru jmen je zastíněna stejně pojmenovanou složkou jiného prostoru jmen. Je obdobou unárního rozlišovacího operátoru `::` v C++.

Příklad

```
namespace GlobalniProstorJmen
{
   class System { }
   class Program
   {
      static void Main(string[] args)
      {
         // System.Console.WriteLine("Text"); // Chyba
         global::System.Console.WriteLine("Text");
         global::System.Console.ReadKey();
      }
   }
}
```

Pokud je např. deklarován alias `SysText` pro prostor jmen `System.Text`

```
using SysText = System.Text;
```

lze dále kvalifikovat třídu `StringBuilder` tímto aliasem a operátorem `::` nebo operátorem tečka

```
SysText::StringBuilder s = new SysText::StringBuilder("text");
SysText.StringBuilder s = new SysText.StringBuilder("text");
```

DATOVÉ TYPY

Jazyk C# rozeznává čtyři druhy datových typů:

- hodnotové,
- referenční,
- ukazatele – v nezabezpečeném kódu deklarovaném pomocí klíčového slova `unsafe`,
- typový parametr – parametr generického typu.

Všechny datové typy mají jako svého základního předka třídu `object`.

Hodnotové typy

Hodnotové typy jsou:

- struktury,
- výčtové typy.

Struktury jsou:

- číselné typy (angl. numeric types):
  - integrální typy (angl. integral types) – celá čísla, znaky
  - reálné typy (angl. floating-point types) – reálná čísla
  - typ `decimal` – desítková čísla,
- typ `bool` – logické hodnoty,
- nulovatelné typy (angl. nullable types) – typy, které mohou obsahovat `null`,
- uživatelem definované struktury.

Celá čísla

Jazyk C# obsahuje 8 celočíselných typů.

Typ	Struktura	Velikost [byte]	Rozsah
`sbyte`	`System.SByte`	1	–128 až 127
`byte`	`System.Byte`	1	0 to 255
`short`	`System.Int16`	2	–32,768 až 32,767
`ushort`	`System.UInt16`	2	0 až 65,535
`int`	`System.Int32`	4	–2,147,483,648 až 2,147,483,647
`uint`	`System.UInt32`	4	0 až 4,294,967,295
`long`	`System.Int64`	8	–9,223,372,036,854,775,808 až 9,223,372,036,854,775,807
`ulong`	`System.UInt64`	8	0 až 18,446,744,073,709,551,615

Uvedené názvy typů jsou jediné možné, tj. narozdíl od C++ nelze psát např. `long int`.

Všechny uvedené typy jsou ve skutečnosti struktury, které jsou deklarované v prostoru jmen `System`. Následující deklarace znamenají totéž:

```
System.Int32 i = 10;
int i = 10;
```

Celočíselné konstanty lze zapsat v desítkové nebo šestnáctkové soustavě ve stejném tvaru jako v C++. Narozdíl od C++ nelze zapsat konstantu v osmičkové soustavě. Celočíselná konstanta je prvního z typů `int`, `uint`, `long` nebo `ulong`, do kterého se vejde.

Explicitní stanovení typu konstanty – pomocí přípony za konstantou.

Přípona	**Typ**
`U` nebo `u`	`uint`
`L` nebo `l`	`long`
`UL` nebo `ul` nebo `LU` nebo `lu`	`ulong`

Rozšiřující konverze, tj. konverze, při nichž nemůže dojít ke ztrátě informace, lze provést automaticky. tj. Např. hodnotu typu `short` lze přiřadit do proměnné typu `int`.

Zužující konverze, tj. konverze, při nichž může dojít ke ztrátě informace, je nutno explicitně předepsat. K tomu slouží operátor přetypování (typ), známý z C++.

Jakýkoliv celočíselný typ lze implicitně konvertovat na reálný typ a typ `decimal`. Opačně se musí použít explicitní konverze.

Např.

```
int i;
long j;
j = i; // OK
i = j; // Chyba
i = (int)j; // OK
```

Celá čísla nelze použít jako logické hodnoty.

Pro celá čísla jsou definovány následující operátory, mající stejný význam jako v C++:

- unární aritmetické operátory `+`, `–`
- binární aritmetické operátory `+`, `-`, `*`, `/`, `%`
- operátory inkrementace a dekrementace `++` a `--`
- relační operátory `<`, `>`, `<=`, `>=`, `==`, `!=`
- bitové operátory `|`, `&`, `~`, `<<`, `>>`, `^`

Při operacích s celými čísly může vzniknout výsledek, který přesahuje rozsah daného typu – může dojít k celočíselnému přetečení. Ke kontrole přetečení existují operátory `checked` a `unchecked`.

Syntaxe:

použití operátorů checked a unchecked:

checked (*výraz* **)**

unchecked (*výraz* **)**

Implicitně se celočíselné přetečení nekontroluje. Zapnutí kontroly celočíselného přetečení v celém programu: menu **Projet | Properties | Build**, tlačítko **Advanced**, zaškrtávací políčko **Check for arithmetic overflow/underflow**.

Výraz v operátoru `checked` bude program za běhu kontrolovat na přetečení. Pokud k němu dojde, vyvolá výjimku `System.OverflowException`.

Výraz v operátoru `unchecked` nebude program za běhu kontrolovat na přetečení.

Pokud se má kontrolovat resp. nekontrolovat přetečení ve skupině výrazů, lze použít příkaz `checked` resp. `unchecked`.

Syntaxe:

příkazy checked a unchecked:

checked { *příkazy* **}**

unchecked { *příkazy* **}**

Příklad

```
class Program
{
   static void Main(string[] args)
   {
      int i = 21; // číslo 21 a vyšší vyvolá výjimku přetečení
      Console.WriteLine("Faktoriál " + i + " je " +
         Matematika.Faktoriál(i));
      Console.ReadKey();
   }
}

class Matematika
{
   public static long Faktoriál(int n)
   {
      long s = 1;
      // Varianta 1
      // while (n > 0) s = checked(s*n--);

      // Varianta 2
      checked {
         while (n > 0) s *= n--;
      }
      return s;
   }
}
```