Platforma .NET

Radek Janoštík

Univerzita Palackého v Olomouci

24. 9. 2024

Úvod

- Předmět (volně) navazuje na Jazyk C# 1 a 2 (KMI/JCS1 a KMI/JCS2)
- Tři hodiny týdně $\rightarrow$ výklad + samostatná práce
- Důraz na **samostatnou** práci

Konzultace, kontakt

- Email: radek.janostik@upol.cz
- Pracovna: 5.073
- Telefon: 585 634 711
- Web: `https://apollo.inf.upol.cz/~janostik/`
- Konzultace: Úterý 11:30 – 13:00 nebo dohodou

Podmínky zápočtu

- Účast na semináři není povinná
- Z každého semináře bude úkol – **samostatná práce**
- Úplné splnění úkolu na semináři $\Rightarrow$ 5 bodů
- Úplné splnění úkolu do 23:59:59 olomouckého času v neděli před následujícím seminářem $\Rightarrow$ 4 body.
- Chyby, neúplnost, „bad practice“ budou penalizovány

Podmínky zápočtu

- Účast na semináři není povinná
- Z každého semináře bude úkol – **samostatná práce**
- Úplné splnění úkolu na semináři $\Rightarrow$ 5 bodů
- Úplné splnění úkolu do 23:59:59 olomouckého času v neděli před následujícím seminářem $\Rightarrow$ 4 body.
- Chyby, neúplnost, „bad practice“ budou penalizovány
- Celkem x úkolů $\Rightarrow$ maximálně $5x$ bodů, **pro zápočet potřeba** $\geq 3.0x$ **bodů**
- $x \approx 11$, tedy $3.0x11 = 33$
- Bude větší úkol za 8 bodů, a ten je nutné splnit alespoň na 6 bodů. (možné opravy)

Podmínky zápočtu

- Účast na semináři není povinná
- Z každého semináře bude úkol – **samostatná práce**
- Úplné splnění úkolu na semináři $\Rightarrow$ 5 bodů
- Úplné splnění úkolu do 23:59:59 olomouckého času v neděli před následujícím seminářem $\Rightarrow$ 4 body.
- Chyby, neúplnost, „bad practice“ budou penalizovány
- Celkem x úkolů $\Rightarrow$ maximálně $5x$ bodů, **pro zápočet potřeba** $\geq 3.0x$ **bodů**
- $x \approx 11$, tedy $3.0x11 = 33$
- Bude větší úkol za 8 bodů, a ten je nutné splnit alespoň na 6 bodů. (možné opravy)
- Alternativa 1: Menší projekt zahrnující probranou látku odevzdaný do měsíce od schválení

Podmínky zápočtu

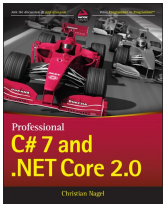
- Účast na semináři není povinná
- Z každého semináře bude úkol – **samostatná práce**
- Úplné splnění úkolu na semináři $\Rightarrow$ 5 bodů
- Úplné splnění úkolu do 23:59:59 olomouckého času v neděli před následujícím seminářem $\Rightarrow$ 4 body.
- Chyby, neúplnost, „bad practice“ budou penalizovány
- Celkem x úkolů $\Rightarrow$ maximálně $5x$ bodů, **pro zápočet potřeba** $\geq 3.0x$ **bodů**
- $x \approx 11$, tedy $3.0x11 = 33$
- Bude větší úkol za 8 bodů, a ten je nutné splnit alespoň na 6 bodů. (možné opravy)
- Alternativa 1: Menší projekt zahrnující probranou látku odevzdaný do měsíce od schválení
- Alternativa 2: Představení některé části .NET platformy, odpřednášení, vymyšlení úkolu pro ostatní

Odevzdávání úkolů

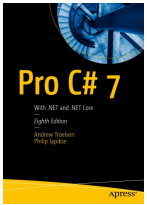
- Osobně na cvičení $\Rightarrow$ 5 bodů
- Emailem $\Rightarrow$ 4 body
  - Emailem na: radek.janostik@upol.cz
  - Předmět: PNE - úkol č. n – př.: "`PNE - úkol č. 1`"
  - Tělo: klidně prázdné, případně nějaké doplnění
  - Příloha: **zip** archiv celého projektu **bez adresářů bin a obj** (antispam), pojmenovaný vaším příjmením bez diakritiky. Př.: `janostik.zip`
  - Přijetí emailu do výše zmíněného termínu před následujícím seminářem
- Nedodržení tvaru předmětu, pojmenování zip archivu $==$ **neodevzdání úkolu**

Doporučená literatura (1/2)

- Christian Nagel. Professional C# 7 and .net Core 2.0. 2018. ISBN: 1119449278,9781119449270

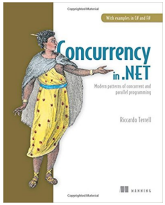


- Andrew Troelsen, Philip Japikse. Pro C# 7: With .NET and .NET Core. 2017, ISBN: 1484230175

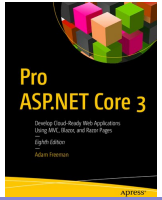


Doporučená literatura (2/2)

- Riccardo Terrell. Concurrency in .NET: Modern patterns of concurrent and parallel programming. 2018. ISBN: 9781617292996



- Adam Freeman. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. 2020, ISBN: 9781484254400,1484254406



Jízdní řád (1/2)

- 24.9.2024 – Úvodní hodina, Vývojové prostředí, jazyky, platforma .NET, Opakování C# (shrnutí minulých semestrů)
- 1. 10. 2024 – Unmanaged prostředí, CLI – MSIL
- 8. 10. 2024 – Unit testy, TDD
- 15. 10. 2024 – Paralelní zpracování dat – vlákna, ThreadPool, optimalizace, PLINQ
- 22. 10. 2024 – F#
- 29. 10. 2024 – F#
- 5. 11. 2024 – ORM

Jízdní řád (2/2)

- 12.11.2023 – ASP .NET (větší úkol)
- 19. 11. 2023 – ASP .NET – konzultace
- 26. 11. 2023 – Blazor, WASM
- 3. 12. 2023 – .NET a Docker, Azure, Cloud – někdo progresivní na *Alternativu 2*?
- 10. 12. 2023 – (umírající) Xamarin? Živořící MAUI?
- 17. 12. 2023 – Rezerva
- Změny v plánu vyhrazeny

Anketa

- Prezenčka

Anketa

- Prezenčka
- Kdo dělal bakalářskou práci na platformě .NET?

Anketa

- Prezenčka
- Kdo dělal bakalářskou práci na platformě .NET?
- C# vs. Java?

Anketa

- Prezenčka
- Kdo dělal bakalářskou práci na platformě .NET?
- C# vs. Java?
- Pracovní zkušenosti s .NET?

Anketa

- Prezenčka
- Kdo dělal bakalářskou práci na platformě .NET?
- C# vs. Java?
- Pracovní zkušenosti s .NET?
- Kam směřuje vývoj vývoje aplikací?

Anketa

- Prezenčka
- Kdo dělal bakalářskou práci na platformě .NET?
- C# vs. Java?
- Pracovní zkušenosti s .NET?
- Kam směřuje vývoj vývoje aplikací?
- Sledujete novinky a vývoj vaší oblíbené platformy?
  - Kde?

Anketa

- Prezenčka
- Kdo dělal bakalářskou práci na platformě .NET?
- C# vs. Java?
- Pracovní zkušenosti s .NET?
- Kam směřuje vývoj vývoje aplikací?
- Sledujete novinky a vývoj vaší oblíbené platformy?
  - Kde?
- Multiplaformita

MS Visual Studio

- Domovské vývojové prostředí pro platformu .NET
- Community version volně dostupná:
  `https://visualstudio.microsoft.com/cs/`
- Vyšší verze by měly být pro studenty zdarma (častá změna jména služby)
- Pro Windows i MacOS
- Neplést s Visual Studio Code
- Dostupné snad všechny funkce, vlastnosti, nejnovější verze .NET Frameworku a .NET Core

Visual Studio Code

- Otevřený, multiplatformní, rozšiřitelný textový editor
- `https://code.visualstudio.com/`
- C# rozšíření pouze pro .NET Core
- Spíše lepší textový editor, „nevodí tolik za ručičku“
- Jak vytvořit konzolovou aplikaci:
  - Nainstalovat C# rozšíření
  - Open Folder – zvolit složku
  - Terminal → New terminal
  - Spustit .NET Core příkaz: `dotnet new console` (vytvoří nový konzolový projekt)
  - Při prvním spuštění doinstaluje závislosti
  - Pak už jde normálně debugovat a pracovat

JetBrains Rider

- Multiplatformní IDE pro C# – `https://www.jetbrains.com/rider/`
- Komplexní a funkční IDE pro .NET Core, vhodná alternativa k Visual Studiu
- Bratříček oblíbené *InteliJ IDEA*
- Osobně jej používám na Linuxu
- Pro studenty zdarma (ověření přes školní email / ISIC)
- Zde na učebnách máme 20 souběžných licencí

Jazyky platformy .NET

- Hlavní jazyk: C# (netřeba představovat)

Jazyky platformy .NET

- Hlavní jazyk: C# (netřeba představovat)
- F#

Jazyky platformy .NET

- Hlavní jazyk: C# (netřeba představovat)
- F#
  - Primárně funkcionální jazyk, lže však i OOP či imperativní
  - Více na `https://docs.microsoft.com/cs-cz/dotnet/fsharp/language-reference/`
  - Budou mu věnovány dva semináře

Jazyky platformy .NET

- Hlavní jazyk: C# (netřeba představovat)
- F#
  - Primárně funkcionální jazyk, lže však i OOP či imperativní
  - Více na `https://docs.microsoft.com/cs-cz/dotnet/fsharp/language-reference/`
  - Budou mu věnovány dva semináře
- Visual Basic .NET
  - Kdo v něm někdy psal?
  - Zjednodušená syntax
  - Nebude zde probírán

Jazyky platformy .NET

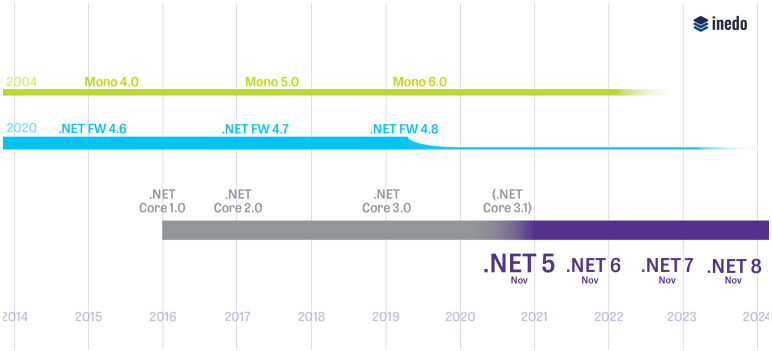
- Hlavní jazyk: C# (netřeba představovat)
- F#
  - Primárně funkcionální jazyk, lže však i OOP či imperativní
  - Více na `https://docs.microsoft.com/cs-cz/dotnet/fsharp/language-reference/`
  - Budou mu věnovány dva semináře
- Visual Basic .NET
  - Kdo v něm někdy psal?
  - Zjednodušená syntax
  - Nebude zde probírán
- Všechny jazyky se kompilují do interního „mezijazyka“ `MSIL`
- Dříve se ještě mohlo používat Visual C++/CIL, zastaralé

Platforma .NET

- Oficiální (marketingový popis):
  - „.NET is a free, cross-platform, open source developer platform for building many different types of applications.“
  - „With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, and IoT.“
- Zkráceně: Jazyky + Sada knihoven + Sada nástrojů
- Nyní důraz na multiplatformitu
- Aktuální verze .NET 8.0.8 (LTS),
- verze 9 již jako RC (vydání asi listopad 2024)

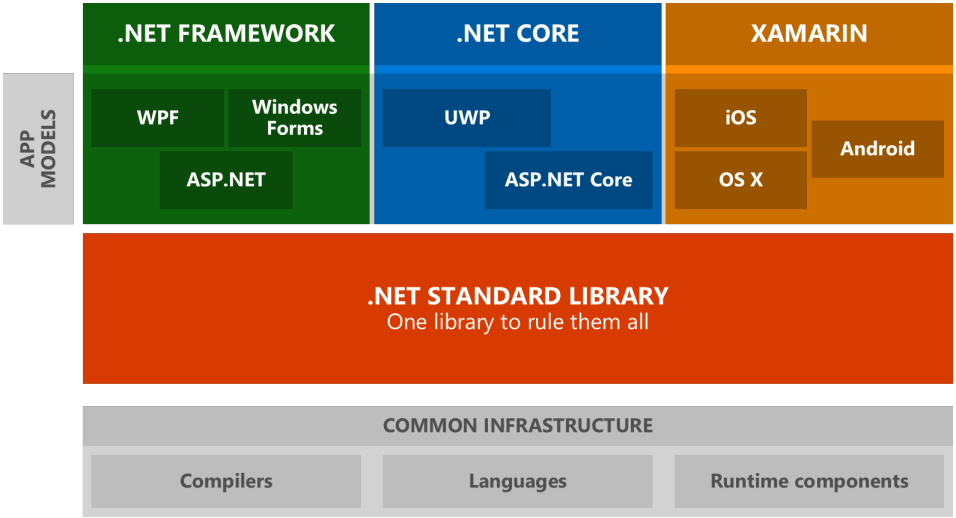
Zmatek ve verzích

- MS opravdu „umí“ pojmenovávat a verzovat produkty



Obrázek: Zdroj: `https://blog.inedo.com/dotnet/net5-release-prep`

Platforma .NET obrázkem



Závěr

- Kvůli předchozí zkrácené definici zde můžeme probrat jakoukoliv zajímavou (rozsáhlejší) knihovnu.
- Nápady vítány (viz Alternativa k zápočtu č. 2)

Opakování – jazyk C#

- Specifikace jazyka: `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/readme`
- Četl někdy někdo?

Proměnné

- Deklarace: `datovýTyp indetifikátor;` př.: `string s;`
- Abychom ji mohli použít ve výrazech, musíme přiřadit hodnotu operátorem `=`
- `s = "Nějaký řetězec;`, jde to i zároveň: `string s = "Nějaký řetězec"`
- Proměnné stejného typu můžeme inicializovat na stejném řádku:
- `int i = 42, j = 0;`
- Automatická inference typu: `var π = 3.14;` proměnná π bude typu `Double`
  - Proměnná musí být inicializována (kompilátor by nepoznal typ)
  - Nesmí se přiřazovat `null`
- Sice .NET podporuje Unicode, ale asi není dobrý nápad používat π jako název proměnné
- Konstanty: `const int x = 15;` – nejde změnit hodnota, musí být inicializovaný při deklaraci

Operátory

- Podobné jako v C++ či Javě, brainstorming:
- Rozdíl mezi `&` a `&&`
- Co dělá operátor: `^=` ?
- Co je to arita operátorů?
- Co dělá operátor `?.`?
- Co dělá operátor `??` ?
- Priority operátorů (`https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/`)

Řízení toku programu

- Podmíněný příkaz `if` – (snad) netřeba představovat.
- `switch / case` – lze nahradit serií `if`

```
switch (title) {
   case "Bc":
   case "Mgr":
      message = "Hello bro";
      break;
   case "PhD":
   case "doc":
      message = "Good morning";
      break;
   case "prof":
      message = "Good morning, my lord";
      break;
   default:
      message = "";
      break;
}
```

- Za `case` musí být konstanta
- `goto` v C# máme, ale nebudeme používat. (proč?)

Cykly

- `for` cyklus: `for (inicializatory; podminka; interator) { výrazy }` – každá z částí může být prázdná
  - (Většinové) použití: Když dopředu známe počet iterací
- `while` cyklus: `while (podminka) { výrazy }`
- `do...while` cyklus `do { výrazy } while (podmínka)`
  - Rozdíl v době testování podmínky
  - (Většinové) použití: Když dopředu neznáme počet iterací
- `foreach` cyklus pro procházení kolekcí:

```
foreach (int number in numbers) {
   Console.WriteLine(number);
}
```

- `break` a `continue`

Pole

- Pole je datová struktura, která obsahuje několik elementů stejného datového typu
- Fixní délka!
- Deklarace, inicializace a přiřazení hodnot: `int[] pole = new int[] {2,4,6};`
- Indexováno od nuly, přístup pomocí indexeru `pole[2]`
- Více dimenzionální(rank) pole: `int[,,] pole3d = new int[3,6,9];`
- Přístup pomocí indexeru s čárkou: `int[2,1]`
- Pouze pro „obdélníková pole“. Kolik může být dimenzí?
- „neobdélníková pole“ jako pole polí: `int[][] pole2d = new int[3][];`

Kolekce

- Systém rozhraní pro práci s „kolekcemi“ proměnlivé délky
- `IEnumerable` – pouze pro procházení (nutné pro `foreach`)
- `ICollection` – zjištění počtu prvků, přidání, odebrání, kopie do pole
- `IList` – K prvkům lze přistupovat na základě pořadí, mají indexer
- Další viz `System.Collections`
- Seznam, spojový seznam, setřízený seznam, fronta, zásobník, množiny, slovník, . . .

Generické typy

- Pomocí generiky můžeme vytvářet metody a třídy, které nejsou závislé na konkrétním typu
- Př.: Chceme implementovat spojový seznam s číselnými hodnotami a to samé pro řetězce
- Bez generiky bychom stejný kód psali 2x jen se změnou typu

```
public class LinkedListNode<T> {
   T value;
   LinkedListNode<T> nextNode;
}
```

- Omezení typu `T`:

```
public class LinkedListNode<T> where T: IEnumerable
```

- Lze omezit: `struct, class, IBlah, Bar, new()` pro (po řadě) hodnotový typ, referenční typ, implementuje rozhraní `IBlah`, dědí z `Bar`, má default kontruktor
- Generické metody (návratový typ, argumenty)

Referenční a hodnotové datové typy

- Hodnotové typy – hodnota uložena přímo na zásobníku
  - atomické typy – int, bool, float, double,...
  - životnost pouze v aktuálním prostředí
  - jako parametr funkce je kopie hodnoty
- Referenční typy – na zásobníku uložena pouze adresa objektu na haldě
  - paměť na haldě přidělena až po použití operátoru `new`, jinak `null`
  - životnost určuje garbage collector
  - jako parametr funkce je kopie adresy
- Klíčové slovo `ref`, `out`

Struktury

- „Odlehčené objekty bez dědičnosti“ – *většinou* uložené na zásobníku
- Čistě pro uložení dat, bez dědičnosti, může implementovat rozhraní
- Hlavní rozdíl oproti třídám: struktury jsou hodnotový datový typ!
- Odlehčení spočívá ve výkonu zásobníku (vs. garbage collector)

```
public struct StagStudent {
    public StagStudent(string name, short year) {
        Name = name;
        Year = year;
    }
    public string Name { get; }
    public short Year { get; }
    public void doNothing(int arg, string arg2) {
    }
}
```

Výčtové datové typy

- `enum` je uživatelsky definovaný datový typ, který může nabývat *n* konstantních hodnot
- Defaultně je na pozadí `int` (lze změnit), číslováno inkrementálně od 0

```
public enum ChessPiece : short {
   King = 6, Queen = 3, Rook = 6, ...
}
```

- Pokud může nabývat více hodnot současně, atribut `[flags]`:

```
[Flags]
public enum FilePermission {
   Execute = 1, Write = 0x2, Read = 0x4
}
```

- Poté může nabývat více hodnot pomocí logického `OR`

```
FilePermission userPerm = FilePermission.Execute | FilePermission.Read;
```

Přetížení (překrytí) operátorů

- V C# je možné definovat operátory na vlastní objekty
- Lze: `+, -, !, ~ , ++, --, true, false, *, /, %, &, |, ^ , <<, >>`
- Lze párově: $==, !=, <, >, <=, >=$
- Nelze ale využívá předchozí: $+=, -=, *=, /=$, %=, &=, $|=$, $\hat{}$ =, $<<=, >>=$
- Nelze: =, ., ?:, ??, $->, =>$, f(x), as, checked, unchecked, default, delegate, is, new, sizeof, typeof

```
public static Complex operator +(Complex c1, Complex c2)
{
return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
}
```

- Pouze syntaktický cukr. Kdo viděl větší použití?

Vlákna

- Objektový mechanismus pro paralelní výpočty, třída `System.Threading.Thread`
- Konstruktor má parametr delegáta na metodu vracející `void`

```
public delegate void ThreadStart();
Thread worker = new Thread(RunFactorial);
worker.Start(param);
```

- Počkání na dokončení činnosti – `worker.Join();`
- Synchronizace:
  - Při přístupu ke sdíleným proměnným více vlákny může dojít k nechtěnému stavu
  - Problém atomicity operací: $x = x + 5$ vnitřně není jedna operace
  - Příkaz `lock (x) { }` zapouzdří objekt x do kritické sekce (vzájemné vyloučení)
  - Pozor na zamykání na `string`

Práce se soubory

- Třídy `DirectoryInfo, FileInfo, DriveInfo`
- Textové čtení celého souboru:

```
FileInfo f = new FileInfo(@"E:\1\test.txt");
if (f.Exists){
   string[] array = File.ReadAllLines(f.FullName);
}
```

- Streamy – obecný mechanismus pro čtení/zápis dat „odněkud někam“

```
FileInfo f = new FileInfo(@"E:\1\test.txt");
if (f.Exists) {
   StreamReader sr = null;
   try {
      sr = f.OpenText();
      string s = "";
      while ((s = sr.ReadLine()) != null) {
         Console.WriteLine(s);
      }}}
```

- Podobně pro zápis. Ošetřovat výjimkami!

Čtení XML

- Práce s DOM (Document Object Model)
- „Stromový přístup“

```
string filePath = @"E:\5\in.xml";
XmlDocument doc = new XmlDocument();
doc.Load(filePath); // Nacteni dokumentu
XmlNodeList nodeList = doc.GetElementsByTagName("item"); // Vybrani elementu "item"
foreach (XmlNode node in nodeList) {
  Console.WriteLine(node.Name);
  Console.WriteLine("Data:");
  foreach (XmlNode child in node.ChildNodes) { // Potomci elementu
    Console.Write(child.Name);
    Console.WriteLine(child.Value);
  }
  Console.WriteLine("Atributy: ");
  foreach (XmlAttribute att in node.Attributes) { // Atributy elementu
    Console.Write(att.Name + ": ");
    Console.WriteLine(att.Value);
  }
}
```

Zápis XML

- Podobný přístup - budování stromu

```
XmlDocument doc = new XmlDocument();
XmlDeclaration xmlDeclaration = doc.CreateXmlDeclaration("1.0", "UTF-8", null);
XmlElement root = doc.DocumentElement;
doc.InsertBefore(xmlDeclaration, root);

XmlElement item = doc.CreateElement("item");
item.SetAttribute("att1", "some value");
item.SetAttribute("att2", "another value");

XmlElement number = doc.CreateElement("number");
number.InnerText = 8876.ToString();
item.AppendChild(number);

XmlElement str = doc.CreateElement("str");
str.InnerText = "Yet another string";
item.AppendChild(str);

doc.AppendChild(item);
XmlTextWriter xmlTextWriter = new XmlTextWriter(@"E:\5\out.xml", Encoding.UTF8);
xmlTextWriter.Formatting = Formatting.Indented;
doc.WriteContentTo(xmlTextWriter);
xmlTextWriter.Close();
```

Serializace/Deserializace

- Ruční zápis a načítání je zdlouhavé a nudné
- Vytvoříme si novou třídu modelující `item` z XML

```
public class Item {
   public int Number { get; set; }
   public string Str { get; set; }
   public int[] Numbers { get; set; }
}
```

- Serializace:

```
Item i = new Item() { Number = 42, Str = "abcdefg", Numbers = new int[]{ 5, 4, 3, 2, 1 }
TextWriter tw = new StreamWriter(@"E:\5\serialize.xml");
XmlSerializer sr = new XmlSerializer(typeof(Item));
sr.Serialize(tw, i);
tw.Close();
```

- Deserializace

```
FileStream f = new FileStream(@"E:\5\serialize.xml", FileMode.Open);
XmlSerializer serializer = new XmlSerializer(typeof(Item));
Item item = (Item)serializer.Deserialize(f);
Console.WriteLine(item);
f.Close();
```

- Jednoduché a bez práce (jednoduchá změna formátu)

Lambda výrazy

- = anonymní funkce pro vytváření delegátů či výrazových stromů

```
delegate int somedelegate(int blah);
static void Main(string[] args) {
   somedelegate del = (i) => i + 5;
}

Func <int, int> fce = ((int i) => i + 2);
Console.WriteLine(fce.Invoke(5));
```

- Mohou být předány jako parametr metody
- Mohou být vráceny metodou
- Při přiřazení však staticky typované

LINQ (1/2)

- Zkratka pro Language INtegrated Query
- Jednotný přístup k datům
- Typová kontrola, našeptávače
- Přístup např. ke:
  - Kolekcím (vše co splňuje IEnumerable)
  - XML
  - Databázovým objektům
  - Webovým službám
- Rozšiřitelné
- Fluent API:

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7 };
IEnumerable<int> bigNumbers2 = numbers.Where(p => p > 5);
```

LINQ (2/2) – dotazovací operace

- Restrikce

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7 };
IEnumerable<int> bigNumbers2 = numbers.Where(p => p > 5);
```

- Selekce

```
List<Person> persons = new List<Person>();
var names = persons.Select(p => new { p.Name, p.Surname });
```

- Řazení

```
persons.OrderBy(p => p.Age).ThenBy(p => p.Name);
persons.OrderByDescending(p => p.Surname).ThenBy(p => p.Name);
```

- Omezení počtu, přeskočení prvních n

```
persons.OrderBy(p => p.Age).ThenBy(p => p.Name).Skip(20).Take(10);
```

- Unikátnost `persons.Distinct();`
- Obsahuje nějaký prvek s vlastností

```
persons.Any(p=>p.Name="Karel");
```

- Počet prvků s vlastností

```
persons.Count(p => p.Age == 30);
persons.Where(p=>p.Age==30).Count();
```

Opakování – co ještě známe

- Práce se SQL databází, ORM
- ASP.NET, MVC, WebAPI
- WPF – GUI, kreslení
- ...

Úkol

- Komplexní prověření znalostí C# a práce s dokumentací
- Programově stáhnout dataset Flags z UC Irvine Machine Learning Repository:
  - ZIP s daty: `https://archive.ics.uci.edu/static/public/40/flags.zip`
  - Popis dat v souboru `flag.names` nebo na `https://archive.ics.uci.edu/dataset/40/flags`
- Načíst dané csv(`flag.data`) a vhodně navrhnout interní programovou datovou strukturu
- Zjistit, které španělsky mluvící země nemají na vlajce červenou barvu
- Zjistit, které země mají horizontální pruhy, ale neobsahují bílou barvu
- Dané země setřídit sestupně podle počtu pruhů
- `JSON` obsahující jméno země, počet pruhů, počet barev na vlajce a počet obyvatel odeslat (opět programově) na email: `radek.janostik@upol.cz` s předmětem `PNE -- výsledky -- odesílatel`
- (Poté klasicky poslat/ukázat zdrojové kódy)
- Bonusový bod: Vypracovat úkol ve VisualBasicu