# 5 Práca s dátami

Pokročilé objektové technológie



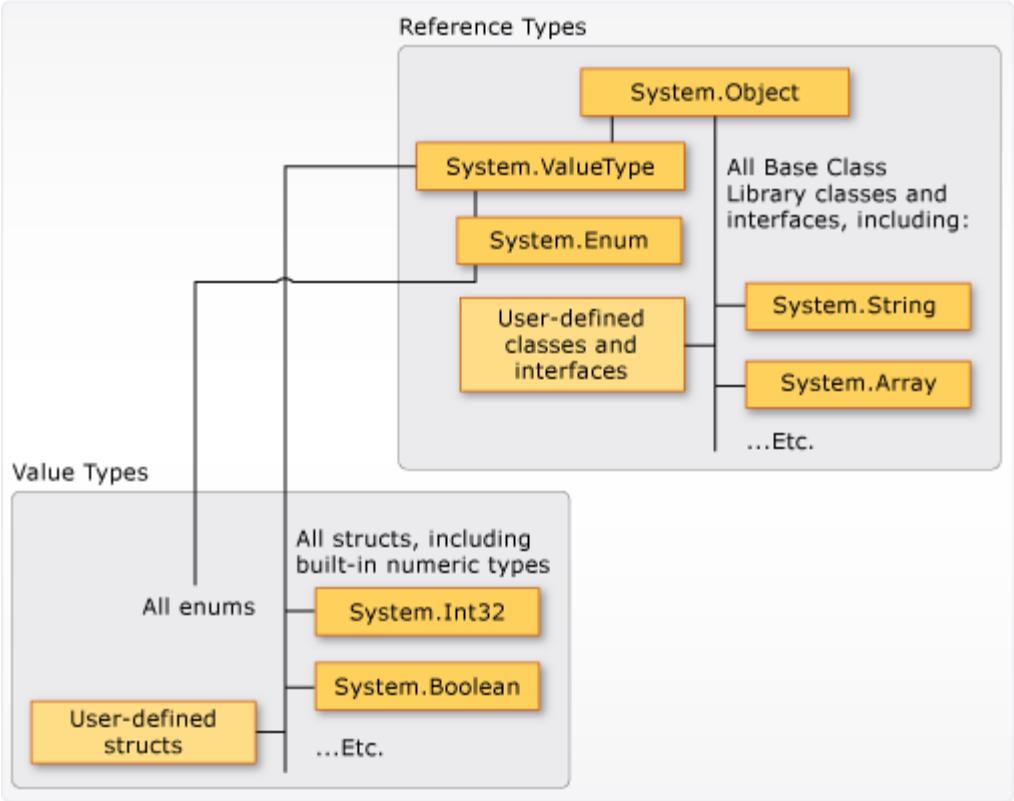
KST Katedra softvérových technológií
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

# Obsah

- Kolekcie
- LINQ
- Serializácia
- Práca so súbormi
- Regulárne výrazy

# Zopakovanie – typy v C#

- **Hodnotové typy** (value types)
  - **struct** (číselné typy, bool, štruktúry), **enum** (vymenované typy)
- **Odkazové typy** (reference types)
  - **class** (vstavané typy: object, string, dynamic), **interface**, **delegate**
- **Smerníkové typy** (pointer types)
  - * (unsafe)




http://msdn.microsoft.com/en-us/library/ms173104.aspx

# Polia a kolekcie

- **Pole**
  - **System.Array** – základná trieda reprezentujúca pole
  - Deklarácia: **type[]** jednorozmerné**; type[,]** dvojrozmerné**, type[][]** rozoklané

- **Kolekcie**
  - Príklady: **hash tabuľka, fronta** (queue), **zásobník** (stack), **slovník** (dictionary), **zoznamy** (list)
  - Umiestnené v menných priestoroch:
    - **System.Collections** – obecné (negenerické) typy, ak nie je potrebné, nepoužívať! Namiesto nich používať typy z nasledujúcich menných priestorov:
    - **System.Collections.Generic**
    - **System.Collections.Concurrent**

# Polia - príklady

```
int[] array1 = new int[5]; // Deklarácia jednorozmerného poľa
int[] array2 = new int[] { 1, 3, 5, 7, 9 }; // Deklarácia a inicializácia 1-rozmerného poľa
int[] array3 = { 1, 2, 3, 4, 5, 6 }; // Alternatívna syntax deklarácie a inicializácie

int[,] multiDimensionalArray1 = new int[2, 3]; // Deklarácia dvojrozmerného poľa
int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } }; // Deklarácia a inicializácia
dvojrozmerného poľa

int[][] jaggedArray = new int[6][]; // Deklarácia rozoklaného poľa
jaggedArray[0] = new int[4] { 1, 2, 3, 4 }; // Nastavenie hodnôt prvého poľa
```

# Kolekcie

| Generikum | Doteraz | Význam |
|---|---|---|
| `Dictionary<TKey, TValue>` | `Hashtable` | Kolekcia dvojíc kľúč-hodnota |
| `List<T>` | `ArrayList` | Dynamické pole |
| `Queue<T>` | `Queue` | FIFO (fronta) |
| `Stack<T>` | `Stack` | LIFO (zásobník) |
| `SortedDictionary<TKey, TValue>` | | Usporiadaná množina dvojíc |
| `SortedList<TKey, TValue>` | `SortedList` | Usporiadaná množina dvojíc |
| `LinkedList<T>` | | Obojsmerný zreťazený zoznam |
| `Collection<T>` | `CollectionBase` | Základná trieda kolekcií |

# Kolekcie – Dictionary príklad

```
var openWith = new Dictionary<string, string>();
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");

openWith["rtf"] = "winword.exe";

string value;
if (openWith.TryGetValue("tif", out value))
    Console.WriteLine("For key = \"tif\", value = {0}.", value);
else
    Console.WriteLine("Key = \"tif\" is not found.");

foreach (var kvp in openWith)
    Console.WriteLine("{0} = {1}", kvp.Key, kvp.Value);
```

# Rozhrania pre enumerovateľné objekty

- IEnumerable, IEnumerable<T>, ICollection, ICollection<T>, IList, IList<T>

- Príklad z definícii typov:
  - public abstract class **Array** : ICloneable, **IList**, IStructuralComparable, IStructuralEquatable { … }
  - public interface **IList** : **ICollection, IEnumerable** { … }
  - public interface **ICollection** : **IEnumerable** { … }
  - public interface **IDictionary : ICollection, IEnumerable** { … }

  - public interface **IList<T>** : **ICollection<T>**, **IEnumerable<T>**, IEnumerable { … }
  - public interface **ICollection<T>** : **IEnumerable<T>**, IEnumerable { … }
  - public interface **IDictionary<TKey, TValue>** : **ICollection<KeyValuePair<TKey, TValue>>**, **IEnumerable<KeyValuePair<TKey, TValue>>**, IEnumerable

# Úvod do LINQu

# Príklad – úloha

```
var people = new List<Person>
{
    new Person("Jozef", "Mrkvicka", Gender.Male, new DateTime(1990, 1, 1)),
    new Person("Frantisek", "Tlsty", Gender.Unknown, new DateTime(1987, 10, 5)),
    new Person("Jan", "Novy", Gender.Male, new DateTime(1972, 8, 6)),
    new Person("Elena", "Chuda", Gender.Female, new DateTime(1991, 4, 2)),
    new Person("Frantiska", "Nova", Gender.Female, new DateTime(1973, 5, 12))
};
```

- **Úloha:** Nájdite všetky osoby, ktorých priezvisko začína na „Nov“ a ktorí sú narodení do roku 1990 vrátane, následne ich utrieďte podľa priezviska

# Príklad – riešenie (1)

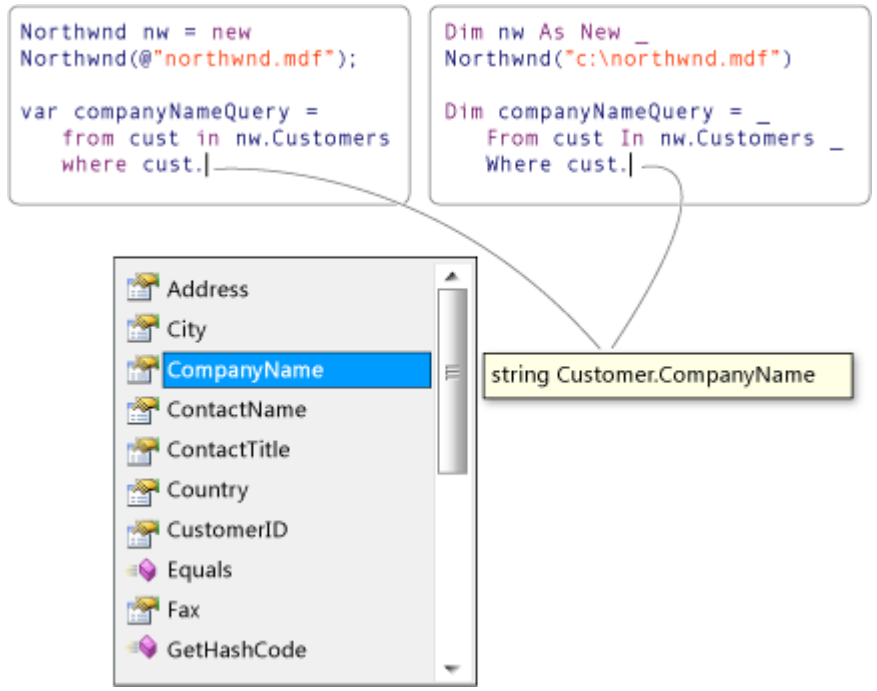
```
var result = new List<Person>();
foreach (var person in people)
{
    if (person.LastName.StartsWith("Nov") && person.Birthdate.Year <= 1990)
        result.Add(person);
}
result.Sort(ComparePersons);

...

private static int ComparePersons(Person p1, Person p2)
{
    return string.Compare(p1.LastName, p2.LastName);
}
```

# Príklad – riešenie (2) – cez LINQ

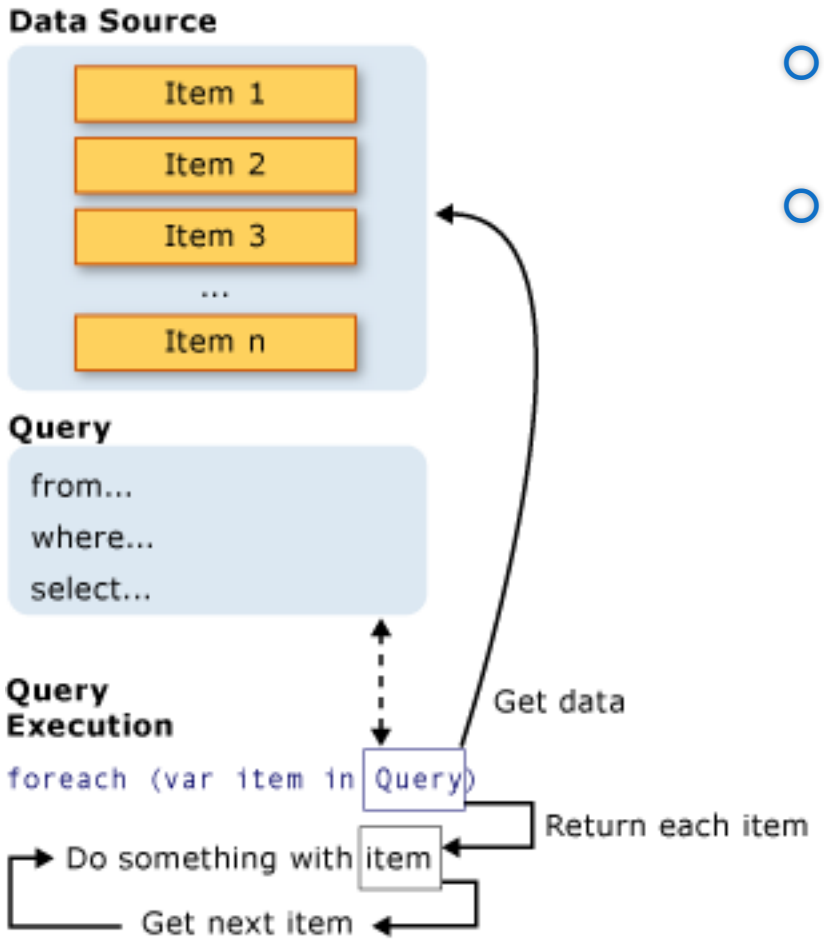
```
var result = from person in people
             where person.LastName.StartsWith("Nov") && person.Birthday.Year <= 1990
             orderby person.LastName
             select person;
```

# LINQ (Language-Integrated Query)

- Rozšírenie jazyka – **dotazovací jazyk**, množinové operácie, transformácie
- Od verzie .NET 3.5
- LINQ to
  - Objects
  - SQL
  - XML
  - Entities
  - DataSet



# LINQ to Objects




- LINQ dotazy nad **kolekciami** implementujúcimi **IEnumerable** alebo **IEnumerable<T>**
- Príklad:

```
// 1. Datovy zdroj
int[] numbers = { 0, 1, 2, 3, 4, 5, 6 };

// 2. Definicia dotazu
var result = from num in numbers
             where num % 2 == 0
             select num;

// 3. Vykonanie dotazu
foreach (var number in result)
{
    Console.WriteLine(number);
}
```

# 1. Dátový zdroj

- Musí byť IEnumerable<T> alebo IQueryable<T>

- Transformácia dát do formy dotazovacieho typu
- XML -> XElement

```
// Create a data source from an XML document.
// using System.Xml.Linq;
XElement contacts = XElement.Load(@"c:\myContactList.xml");
```

- SQL

```
// Create a data source from a SQL Server database.
// using System.Data.Linq;
DataContext db = new DataContext(@"c:\northwind\northwnd.mdf");
```

# 2. Dotaz

- Špecifikuje požadované dáta, ich utriedenie, zoskupenie, ...
- Nová syntax (stručnejší a jednoduchší kód)
- Iba premenná s dotazom

```
// custQuery is an IEnumerable<IGrouping<string, Customer>>
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

# 3. Vykonanie

- Získanie dát: foreach
- **Odložené vykonanie:**
- Možné použitie dotazu viackrát:
  - Budú dáta stále také isté?

```
var evenNumQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

int evenNumCount = evenNumQuery.Count();
```

- **Okamžité vykonanie:**

```
List<int> numQuery2 =
    (from num in numbers
     where (num % 2) == 0
     select num).ToList();
```

# Transformácia dát

- Spojiť viacero zdrojov do nového výstupu
- Výber niektorých vlastností
- Vykonať na nich rôzne operácie
- Zmeniť formát dát

# Dostupné klauzuly

- Operátory selekcie – filtrovania
  - Where
- Operátory projekcie
  - Select, SelectMany
- Triediace operátory
  - OrderBy, OrderByDescending, ThenBy, ThenByDescending
- Množinové operátory
  - Distinct, Union, Intersect, Except
- Agregačné
  - Count, Sum, Min, Max, Average, Aggregate
- Konverzné
  - ToArray, ToList, ToDictionary, OfType

https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

# Príklad – dotaz pre získanie súborov

```
string startFolder = @"C:\Program Files (x86)\Microsoft Visual Studio 12.0\";
var dir = new DirectoryInfo(startFolder);
var fileList = dir.GetFiles("*.*", SearchOption.AllDirectories);

var fileQuery = from file in fileList
                where file.Extension == ".txt"
                orderby file.Name
                select file;

foreach (var fi in fileQuery)
    Console.WriteLine(fi.FullName);

var newestFile = (from file in fileQuery
                  orderby file.CreationTime
                  select new { file.FullName, file.CreationTime }).Last();
```

# Príklad – vytvorenie XML súboru

```
var studentsToXML = new XElement("Root",
    from student in students
    let x = String.Format("{0},{1},{2},{3}", student.Scores[0],
            student.Scores[1], student.Scores[2], student.Scores[3])
    select new XElement("student",
                new XElement("First", student.First),
                new XElement("Last", student.Last),
                new XElement("Scores", x)
            ) // end "student"
        ); // end "Root"
```

# Ako LINQ funguje?

- LINQ dotaz sa prekladá do volania zreťazených rozširujúcich metód IEnumerable (alebo IQueryable):

- Implementácia vďaka pridaným prvkom do jazyka (od .NET 3.0):
  - Rozširujúce (extension) metódy
  - Lambda výrazy
  - Implicitne typové premenné
  - Anonymné typy

```
var result = from n in numbers
             where n % 2 == 0
             orderby n
             select 1.0 / n;
```

```
var result = numbers.Where(n => n % 2 == 0)
                    .OrderBy(n => n)
                    .Select(1.0 / n);
```

# Rozširujúce (extension) metódy

- Pridávajú metódy k existujúcim typom bez vytvárania odvodených typov, rekompilácie alebo modifikácie existujúcich typov

```
public static class MyExtensions
{
    public static int WordCount(this String str)
    {
        return str.Split(new char[] { ' ', '.', '?' },
                         StringSplitOptions.RemoveEmptyEntries)
                  .Length;
    }
}
```

# Rozširujúce (extension) metódy

- Definovaná ako statická metóda v negenerickej statickej triede
- Volané ako inštančná metóda
- Prvý parameter (typ, ktorý bude rozšírený o novú metódu) musí byť označený kľúčovým slovom this

```
public static class MyExtensions
{
    public static int WordCount(this String str)
    {
        return str.Split(new char[] { ' ', '.', '?' },
                         StringSplitOptions.RemoveEmptyEntries)
                  .Length;
    }
}

...

string s = "Ahoj, ako sa máš?";
int count = s.WordCount(); // count = 4
```

# Rozširujúce (extension) metódy pre LINQ

- Definované metódy v triedach `Enumerable` / `Queryable`

- Príklady niektorých metód v triede Enumerable:
  - `public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate)`
  - `public static IEnumerable<TResult> Select<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, int, TResult> selector)`
  - `public static TSource Max<TSource>(this IEnumerable<TSource> source)`
  - ...

# Lambda výrazy

- Zavedené od C# 3.0
- Nahradzujú anonymné metódy zavedené od C# 2.0 (delegate() { … } )
- Možný tvar lambda výrazov:
  - vstupný parameter => výraz
  - (vstupné parametre) => výraz
  - (vstupné parametre) => { príkazy; }

```
x => x.IsMan  // alebo aj  (x) => x.IsMan

(x, y) => x == y

(int x, string s) => s.Length > x

() => SomeMethod()

n =>
{
   string s = n + " " + "World";
   Console.WriteLine(s);
}
```

# Implicitne typové premenné

- Kľúčové slovo `var`
- Automaticky odvodený typ
- Možné použitie len v lokálnych premenných

```
var result = from person in people
             where person.LastName.StartsWith("Nov") &&
                   person.Birthdate.Year <= 1990
             orderby person.LastName, person.FirstName descending
             select person;
```

Odvodený typ pri kompilácií bude: `IOrderedEnumerable<Person>`

```
result.GetType() // vráti typ objektu za behu programu, t.j.
System.Linq.OrderedEnumerable`2[UkazkyLINQ.Person,System.String]
```

# Anonymné typy

- Poskytujú pohodlný spôsob zapuzdrenia **množiny vlastností (len na čítanie)** do jedného typu bez nutnosti jeho explicitného definovania
- Anonymný typ je generovaný kompilátorom a preto nie je dostupný zo zdrojového kódu
- Typ každej vlastnosti je odvodený kompilátorom

```
var myAnonymouseType = new {FirstName = "Janko",
                            LastName = "Mrkvicka"};
```

# Súbory

# Triedy zastupujúce súbory a adresáre

- `Directory`, `File`
  - Statické metódy
  - Iba na jednu operáciu
- `DirectoryInfo`, `FileInfo`
  - Stavové
  - Na viacej operácii

```
File.Copy(@"C:\Data\MojSubor.txt", @"D:\Data\MojaKopia.txt");

var myFile = new FileInfo(@"C:\Data\MojSubor.txt");
myFile.CopyTo(@"D:\Data\MojaKopia.txt");
```

# Prúdy (streams)

- Hlavné úlohy
  - Čítanie toku dát
  - Zápis toku dát
  - Môže podporovať aj hľadanie – pohyb v toku dát
- Abstrakcia sekvencie bytov
  - Súbor, Pamäť
  - Vstupno-výstupné zariadenie
  - Sieťový protokol, Pipy

# Trieda Stream

- Základná trieda pre všetky streamy
- Nemusí podporovať všetky činnosti
  - CanRead, CanWrite, CanSeek
- Čítanie a zapisovanie – Read, Write
- Hľadanie – Seek, SetLength, Position, Length
- Buffrovanie – Flush
- Uvoľnenie systémových zdrojov - Close

# Stream triedy

- MemoryStream – nad pamäťou
- NetworkStream – nad sieťovým spojením
- CryptoStream – prepája dátový stream s kryptografickou transformáciou
- FileStream – práca so súbormi
- BufferedStream – pridáva „bufrovaciu“ funkcionality iným streamom

# Binárne súbory

- FileStream
- Základné vstupy:
  - Názov súboru
  - Režim otvorenia – FileMode
    - Append, Create, CreateNew, Open, OpenOrCreate, Truncate
  - Režim prístupu – FileAccess
    - Read, ReadWrite, Write
  - Režim zdieľania – FileShare
    - Inheritable, None, Read, ReadWrite, Write

# Binárne súbory

```
// Zapis do binarneho suboru
using (var writer = new BinaryWriter(File.Open(@"D:\data.bin", FileMode.Create)))
{
    writer.Write(firstName);  // string
    writer.Write(lastName);   // string
    writer.Write(isMale);     // bool
    writer.Write(age);        // float
}

// Citanie z binarneho suboru
using (var reader = new BinaryReader(File.Open(@"D:\data.bin", FileMode.Open)))
{
    string firstName = reader.ReadString();
    string lastName = reader.ReadString();
    bool isMale = reader.ReadBoolean();
    float age = reader.ReadSingle();
}
```

# Textové súbory

- Dá sa použiť aj FileStream
- StreamReader a StreamWriter
  - Špeciálne upravené
  - Automaticky rozpoznajú bod zastavenia
  - ReadLine, Write
  - Automatické rozpoznanie kódovania
    - ASCII, Unicode, UTF7, UTF8, BigEndianUnicode
  - Vstupný parameter môže byť aj iný stream

# Textové súbory

```
// Zapis do suboru
using (var sw = new StreamWriter("TestFile.txt"))
{
    sw.Write("This is the ");
    sw.WriteLine("header for the file.");
    sw.Write("The date is: ");
    sw.WriteLine(DateTime.Now);
    sw.WriteLine("I can write ints {0} or floats {1}, and so on.", 1, 4.2);
    sw.Close();
}

// Citanie zo suboru
using (var sr = new StreamReader("TestFile.txt"))
{
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        Console.WriteLine(line);
    }
    sr.Close();
}
```

# Príklad – jednoduché načítanie súboru

```
string text = File.ReadAllText(path);

string[] lines = File.ReadAllLines(path);

byte[] bytes = File.ReadAllBytes(path);
```

# Atribúty (attributes)

- Direktíva prekladača – ovplyvňujú výsledný kód
- Dodatočné informácie do zostavení
- Realizované prostredníctvom tried
- Možnosť definovať aj vlastné

# Atribúty

- Dodatočné informácie o položke
- Pred definíciou položky:
  - [NázovAtribútu(parametre)]
- Použitie nad:
  - Metódami
  - Triedami
  - Argumentami metód
  - Zostaveniami

# Serializácia

- Proces konverzie objektu do stavu:
  - Perzistentného
  - Transportačného

- V .NET:
  - Binárna
  - XML, SOAP
    - Iba „public“ položky
    - Najčastejšie na zdieľanie dát cez web
  - WCF (DataContract)

# Binárna serializácia

- Zdieľanie objektov medzi rôznymi aplikáciami (cez clipboard)
- Objekt serializovať do streamu
  - Disk, pamäť, po sieti
- Uloženie na disk alebo na prenos objektov medzi aplikáciami alebo aplikačnými doménami

# Základná serializácia

- Najjednoduchší spôsob
- Označiť triedu atribútom [Serializable]

```
[Serializable]
public class MyObject {
  public int n1 = 0;
  public int n2 = 0;
  public String str = null;
}

MyObject obj = new MyObject();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";
```

# Základná serializácia

```
// Ulozenie objektu na disk
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Create,
   FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();


// Nacitanie objektu
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Open,
FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();
```

# Selektívna serializácia

- Ak trieda obsahuje zložky, ktoré nechcete serializovať
- Označiť položku [NonSerialized], [OptionalFieldAttribute]

```
[Serializable]
public class MyObject
{
  public int n1;
   [NonSerialized]
  public int n2;
  public String str;
}
```

# Vlastná serializácia

- Úprava procesu serializácie
- Spustenie vlastných metód
  - OnDeserializedAttribute, OnDeserializingAttribute, OnSerializingAttribute, OnSerializedAttribute
- Implementovať ISerializable interface
  - GetObjectData
  - Špeciálny konštruktor
- Volať konštruktor predka
- Zabezpečiť prístup k GetObjectData

# Vlastná serializácia

```
[Serializable]
public class MyObject : ISerializable {
  public int n1;
  public int n2;
  public String str;

  public MyObject() {}

  protected MyObject(SerializationInfo info, StreamingContext
   context) {
    n1 = info.GetInt32("i");
    n2 = info.GetInt32("j");
    str = info.GetString("k");
  }
```

# Vlastná serializácia

```
public virtual void GetObjectData(SerializationInfo info, StreamingContext context)
{
  info.AddValue("i", n1);
  info.AddValue("j", n2);
  info.AddValue("k", str);
}
```

# WCF serializácia

```
[DataContract]
class Person
{
    [DataMember]
    public string FirstName;
    [DataMember]
    public string LastName;
    [DataMember]
    public int ID;
}
```

```
// Zapis osoby do XML
Person person = new Person("Zighetti", "Barbara", 101);
FileStream writer = new FileStream("mvfile.xml",
                                   FileMode.Create);
var ser = new DataContractSerializer(typeof(Person));
ser.WriteObject(writer, person);
writer.Close();

// Nacitanie osoby z XML
FileStream reader = new FileStream("mvfile.xml",
                                   FileMode.Open);
var ser = new DataContractSerializer(typeof(Person));
var person = (Person)ser.ReadObject(reader, true);
reader.Close();
```

# Regulárne výrazy

- Trieda `System.Text.RegularExpressions.Regex`
- Napríklad: `^(?<frame>[0-9]+)-(?<id>[0-9]+).xml$`

- Znaky:
  - `^` začiatok reťazca
  - `$` koniec reťazca
  - `(?<frame>)` pomenovanie skupiny
  - `[]` množina, napr. [A-Za-z0-9]
  - `[^]` záporná množina [^A-Za-z0-9] - záporné je všetko
  - `\` špeciálny znak (escape)
- Výskyty:
  - `.` výskyt 1
  - `?` výskyt 0 alebo 1
  - `*` výskyt 0 alebo viac
  - `+` výskyt 1 alebo viac
  - `{n}` presne n výskytov
  - `{min, max}` minimálne `min` a maximálne `max` výskytov

Zdroj: http://www.codeproject.com/Articles/9099/The-30-Minute-Regex-Tutorial

Ďalšie zdroje: google.com: „C# Regular Expressions Cheat Sheet“

# Regulárne výrazy - príklad

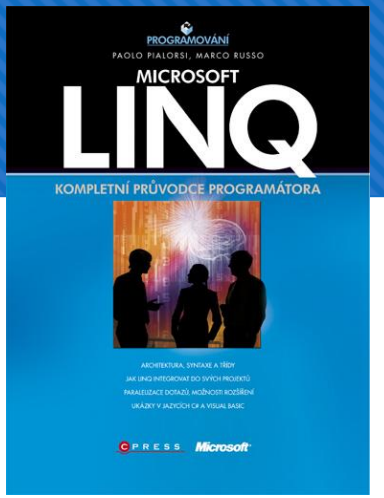
```
string[] addresses = {"127.0.0.1", "255.255.x", "Unicode", "158.193.1.1" };


const string sIpPattern = @"(\d{1,3}\.){3}\d{1,3}";
foreach (string s in addresses)
{
    Console.Write("{0,16}", s);
    if (Regex.IsMatch(s, sIpPattern, RegexOptions.IgnoreCase))
        Console.WriteLine(" is IP address");
    else
        Console.WriteLine();
}
```

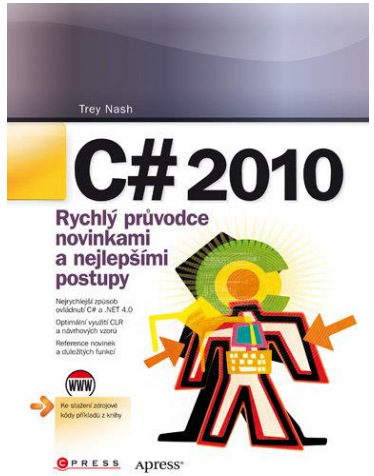
# Literatúra


- Pialorsi, P., Russo, M.: *Microsoft LINQ - Kompletní průvodce programátora*

- Nash, T.: *C# 2010 - Rychlý průvodce novinkami a nejlepšími postupy*

- 101 LINQ príkladov (http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b)

# Ďakujem za pozornosť

Štefan Toth (stefan.toth@fri.uniza.sk)