Nové prvky jazyka C# 2.0

Verze 2006

Aleš Keprt

Ales@Keprt.cz

Katedra informatiky

Univerzita Palackého, Olomouc

Verze jazyka C#

- Specifikace C# není omezena jen na .NET
- Zatím všechny implementace C# jsou v rámci platformy .NET

- Verze:

  C# 1.0 pro .NET 1.0 ➔ Visual C# 2002

  C# 1.1 pro .NET 1.1 ➔ Visual C# 2003

  C# 2.0 pro .NET 2.0 ➔ Visual C# 2005

Třináct nových prvků jazyka C#

1. Generické typy a metody
2. Iterátory
3. Částečně deklarované třídy
4. Nulovatelné typy
5. Bezejmenné metody, vnější proměnné
6. Operátor ::
7. Statické třídy
8. Zástupná jména seskupení
9. Odlišná přístupová práva pro get a set
10. Kovariance a kontravariance
11. Pole pevné délky
12. Spřátelená seskupení
13. Nevypisování varovných hlášení

1. Generické typy a metody

- Umožňují větší rozsah re-use
- Vhodné pro případy, kde se stejný kód opakuje pro různé datové typy
- Generické typy a metody jsou jakési vzory, jejichž instance jsou teprve skutečnými typy a metodami

- Pozn.: typy = třídy, struktury, rozhraní

Příklad – generická funkce

- funkce pro záměnu hodnot dvou proměnných

```
void swap<T>(ref T a, ref T b) {
   T c = a;
   a = b;
   b = c;
}
```

T je jméno typového parametru

- **Použití:**

```
int a,b;
swap<int>(ref a, ref b);
```

Zde jsme za T dosadili konkrétní typ **int**

- **swap(ref a, ref b);**

Typová inference (překladač rozumí, že tam má být **<int>**)

Příklad – jednoduchý zásobník

```
class Stack<T> {
  T[] data;
  int pos;

  public Stack() {
    data = new T[10];
    pos = 0;
  }
```

```
public void push(T value) {
  data[pos++] = value;
  if(pos >= data.Length)
    pos = 0;
}

public T pop() {
  if(pos==0)
    pos = data.Length;
  return data[--pos];
}
```

Příklad – použití zásobníku

```
Stack<string> s = new Stack<string>();
```

- Výhody:
  - Silná typová kontrola
  - Bezpečnější kód
  - Rychlejší kód (u hodnotových typů)
- Praktické důsledky na vývoj softwaru:
  - Méně chyb → Rychlejší vývoj → Levnější vývoj

Kompilace generického kódu

- Kompilace probíhá jinak než v C++
- Každý hodnotový typ má vlastní kód
- Všechny referenční typy sdílí jednu společnou instanci kódu
  - Důvod: Samotná reference je vždy stejná
  - Důsledek: Kratší programy

Omezení typových parametrů

- Základní operace s typovými parametry:
  - Přiřazení
  - Porovnání s **`null`**
  - Volání metod třídy **`System.Object`**
- Ostatní jen s použitím klauzule **`where`**
- Klauzule **`where`** určuje, co vše typový parametr podporuje

`where` – volání metod

- Metody, které chceme volat, umístíme do rozhraní
- Uvedeme požadavek `where T:MyInterface`
- Příklad: Generická metoda `test_any()` volá na svém parametru metodu `test()`:

```
interface ITestable {
  void test();
}

void test_any<T>(T obj) where T : ITestable {
  obj.test();
}
```

`where` – vytváření nových objektů

- Uvedeme požadavek `T:new()`

```
T create<T>() where T:new() {
  return new T();
}
```

- Použití:

```
MyClass a = create<MyClass>();
```

- Platí jen pro konstruktor bez parametrů
- Není nutno používat při vytváření polí

`where` – omezení na pouze hodnotové nebo referenční typy

- Omezení na referenční typy:

```
class RefOnly<T> where T : class {...}
```

  Můžeme pak používat operátor `as`:

```
var as Type
```

- Omezení na hodnotové typy:

```
class ValueOnly<T>where T : struct {...}
```

  Znemožní porovnání s null

  `struct` zastupuje všechny hodnotové typy

`where` – vícenásobná omezení

- Je možno uvádět více typových parametrů
- Lze definovat vztahy mezi parametry
- Příklad ukazuje požadavek na dědičnost:

```
class MyClass<A, B, C>
   where A : B,new()
   where B : C {
}
```

vícenásobná omezení oddělena čárkou

- B a C jsou zde „odhalené“ (naked) parametry
- Mezi dvěma `where` se nepíše žádný oddělovač
- Omezení `new()` musí být v seznamu poslední

Dědičnost – zakázané konstrukce

- Typový parametr nemůže být předkem
- Následující příklad nefunguje:

```
class Outer<T> {
   class Inner : T {}
}
```

Dědičnost – povolené konstrukce

- BaseType nám poslouží jako bázová třída

```
class BaseType { }
```

- Tato generická třída je úplně samostatnou třídou!

```
class BaseType<T> { }
```

- Type1 dědí konkrétní třídu BaseType

```
class Type1 : BaseType { }
```

- Chyba! Nelze zde zjistit typ parametru T

```
class Type2 : BaseType<T> { }
```

- Type3 je generická třída dědící z BaseType<T>

```
class Type3<T> : BaseType<T> { }
```

- Type4 je generická třída dědící z BaseType<int>

```
class Type4<T> : BaseType<int> { }
```

Co všechno může být generické

- Třídy (class)
- Struktury (struct)
- Rozhraní (interface)
- Metody
- Delegáty

- Typový parametr generické třídy lze používat ve všech jejích metodách (viz příklad se zásobníkem)

Výchozí parametry (default)

- Slovo **`default`** zastupuje výchozí hodnotu typu

```
int a = default(int);
MyClass b = default(MyClass);
MyStruct c = default(MyStruct);
```

- Použitelné i mimo generické programování
- Výchozí hodnota třídy je **`null`**
- Výchozí hodnota číselného typu je binární nula
- Výchozí hodnota struktury se zavádí rekurzivně jako výchozí hodnota všech jejích složek

(konec 1.části)

2. Iterátory

- Iterátor je programový blok obsahující příkaz **`yield`**
- Iterátory mají stejný účel jako enumerátory (viz příkaz **`foreach`**)
- Psát iterátory je MNOHEM snazší

Iterátory - příklad

- Přidáme podporu do našeho zásobníku:

```
public IEnumerator<T> GetEnumerator() {
      for(int i=0; i<pos; i++) {
        yield return data[i];
      }
   }
}
```

tento iterátor je generický

zde postupně jakoby „vracíme“ jednotlivé hodnoty

- A to je vše! ☺

Iterátory - implementace

1. Na začátku je volána metoda **`s.GetEnumerator()`**.
2. V místě příkazu **`yield`** je provádění kódu pozastaveno, je uchován kontext a hodnota **`data[i]`** je dosazena do proměnné **`v`**
3. Je provedeno tělo příkazu **`foreach(`***typ* **`v in …)`**.
4. Kontext je přepnut zpět do metody **`GetEnumerator`** a vykonávání pokračuje za příkazem **`yield`** opět až po další příkaz **`yield`** (můžeme tedy také umístit několik **`yield`** na různá místa v metodě).
5. Body 2 až 4 se opakují, dokud neskončí provádění metody **`GetEnumerator`**. Tím je ukončeno i provádění **`foreach`**.

Pojmenovaný iterátor

- Vytvoříme iterátor vracející pro dané N mocniny dvojky od 2^1 až po 2^N.

```
IEnumerable Power(int N) {
   int counter = 0;
   int result = 1;
   while(counter++ < N) {
      result *= 2;
      yield return result;
   }
}
```

jméno iterátoru

Použití iterátorů

- Iterátory se používají stejně jako enumerátory
- Nepojmenovaný iterátor:

```
foreach(string v in mystack) {
   Console.WriteLine(v);
}
```

jméno objektu

- Pojmenovaný iterátor:

```
foreach(int i in Power(10)) {
   Console.WriteLine(i);
}
```

jméno metody

Předčasné ukončení iterace

- Příkaz `yield return x;` vrací další hodnotu
- Příkaz `yield break;` ukončí iteraci (obyčejné `return;` použít nelze)

3. Částečně deklarované třídy

- Třídu lze nyní deklarovat ve více souborech

```
partial class MyClass : BaseClass, Interface1 {
  partial class Inner<T> {
    T a;
  }
}
```

první soubor

```
partial class MyClass : Interface2 {
  partial class Inner<T> {
    T b;
  }
}
```

druhý soubor

Pravidla částečných deklarací

- Je-li třída **`partial`**, musí tak být deklarována ve všech jejích částech
- Všechny části musejí být v jednom modulu
- Jména typů, generických parametrů, přístupová práva a modifikátory musejí ve všech částech souhlasit

---

- Visual Studio 2005 používá tento konstrukt pro oddělení kódu formulářů na část generovanou v IDE a část psanou ručně

4. Nulovatelné typy

- Hodnotové typy nemohou být **`null`**
- Např. při práci s SQL by se to však hodilo

- Nulovatelné typy jsou hodnotové typy, jimž lze přiřadit také **`null`**
- Deklarace: Přidáme otazník za název typu

```
int? x;
Console.WriteLine(x==null ? "null"
                          : x.Value.ToString());
if(x.HasValue)
  Console.WriteLine("x má hodnotu");
```

Implementace nulovatelných typů

- Jsou to instance generického typu
- `T?` je totéž jako `System.Nullable<T>`
- Vlastnost `HasValue` – vrací `true/false`
- Vlastnost `Value` – vyvolá výjimku při `null`
- Konverze `T?` na `T` pouze explicitně
- Konverze `T` na `T?` je implicitní
- `T?` má všechny operátory jako `T`

Nulovatelné typy – operace s null

- Výsledkem operací s `null` je vždy `null`
- Operátor `??` umožní dosadit výchozí hodnotu

```
int? x;
int y = x ?? -1;
```

Je to totéž jako `if(x==null) y=-1; else y=x;`

- Operátor `??` lze používat i bez nulovatelných typů
- Typ bool? je specifický – poskytuje trojhodnotovou logiku (chová se stejně jako SQL typ Boolean)

5. Bezejmenné (anonymní) metody

- Přiblížení funkcionálním jazykům
- Vhodné pro psaní delegátů

```
button.Click += delegate
   {MessageBox.Show("Click!");};
```

Vnější proměnné

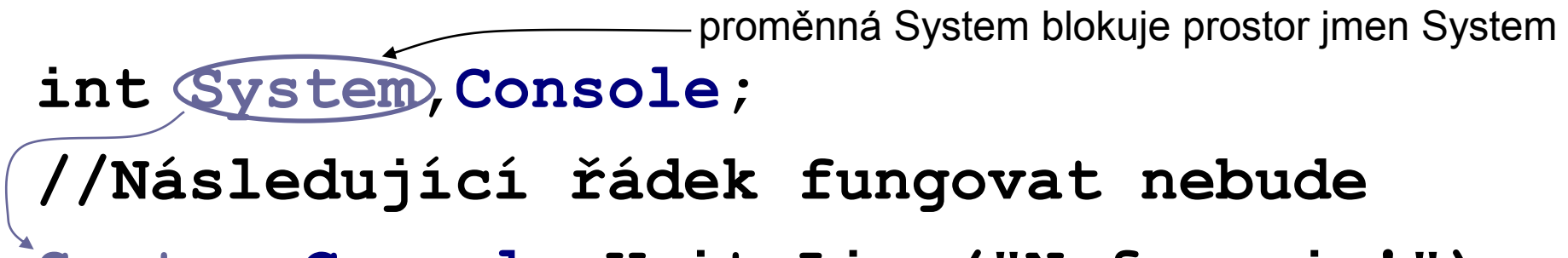
- Bezejmenná metoda je samostatným blokem kódu, ale sdílí proměnné s nadřazeným blokem!

```
int a = 0;
button.Click += delegate
  {MessageBox.Show((++a).ToString());};
a = 23;
```

- V tomto příkladu se všechno odkazuje na jedinou instanci hodnotové proměnné **a**
- Parametry ve formě **ref** a **out** se nesdílí

6. Operátor ::

- Zpřístupňuje (nejen) globální prostor jmen




```
int System,Console;
//Následující řádek fungovat nebude
System.Console.WriteLine("Nefunguje!");
//Použijeme operátor ::
global::System.Console.WriteLine("OK");
```

- Vlevo uvádíme slovo `global` nebo zástupné jméno prostoru jmen (namespace alias)

7. Statické třídy

- Píšeme jednoduše **`static class ...`**
- Statická třída:
  - Má jen statické součásti
  - Nemá konstruktor (statický mít může)
  - Nelze vytvářet instance
  - Je zapečetěná (**`sealed`** – nelze ji dědit)

8. Zástupná jména seskupení

- Potřebujeme odlišit dvě verze stejné komponenty
- Použijeme parametr překladače /r

```
/r:Komponenta_verze1=grid.dll
/r:Komponenta_verze2=grid20.dll
```

- V kódu si tyto komponenty zpřístupníme

```
extern alias Komponenta_verze1;
extern alias Komponenta_verze2;
```

9. Odlišná přístupová práva pro get a set

- Nově mohou mít **`get`** a **`set`** odlišná práva

```
public string Name {
   get {return name;}
   protected set {name = value;}
}
```

public

omezeno na protected

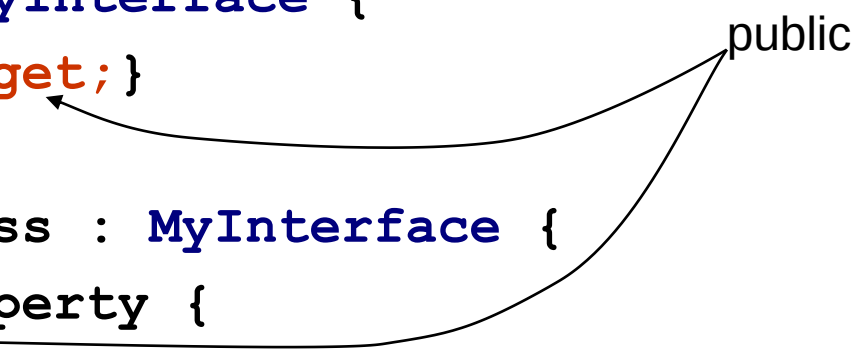
- Pravidla:

1. Vnitřní modifikátor musí být více restriktivní
2. Modifikovat lze jen jedno z get/set, ne obojí

Odlišná přístupová práva pro get a set – Implementace rozhraní

- Implementace rozhraní není děděním
- Akcesor, který není součástí rozhraní, lze doplnit a nastavit mu jiná přístupová práva

```
public interface MyInterface {
  int MyProperty {get;}
}
public class MyClass : MyInterface {
  public int MyProperty {
    get {...}
    protected set {...}
  }
}
```



10. Kovariance a kontravariance

- Obojí se týká delegátů
- Kovariance umožňuje použít metodu, jejíž návratový typ je potomkem návratového typu deklarovaného v delegátu
- Kontravariance umožňuje použít metodu s parametry, které jsou předky parametrů deklarovaných v delegátu

Kovariance a kontravariance příklad

```
class A {}
class B : A {}
class C : B {}
class Test {
  delegate B Metoda(B param);
  B MetodaA(A param) {...}
  C MetodaC(B param) {...}
  public Test() {
    Metoda ma = new Metoda(MetodaA);
    Metoda mc = new Metoda(MetodaC);
  }
}
```

definujeme strom dědičnosti

delegát používá typ B

A místo B: kontravariance

C místo B: kovariance

11. Pole pevné délky

- V **`unsafe`** bloku lze deklarovat pole pevné délky, které je fyzicky uloženo na místě deklarace
- Je to hodnotový typ, neprovádí se kontrola přetečení bufferu atd.

```
struct MyArray {
   unsafe public fixed char path[256];
   private int reserved;
}
```

12. Spřátelená seskupení

- Chceme-li, aby seskupení (assembly) **B** mělo přístup do všech (i neveřejných) součástí seskupení **A**, pak v seskupení **A** použijeme následující atribut:

```
[assembly:InternalsVisibleTo
            ("B",PublicKeyToken="...")]
```

- Public key token je 8bajtová hodnota sloužící k rozlišení dvou stejně pojmenovaných seskupení (od různých autorů apod.)

13. Nevypisování varovných hlášení

- Vypisování varovných hlášení lze ovlivnit přímo v kódu, použitím nové direktivy

```
#pragma warning disable 414,3021
#pragma warning restore 414,3021
```

- Příklad zakáže a pak obnoví varovná hlášení číslo 414 a 3021
- Tento konstrukt nelze použít pro povolení hlášení zakázaných nastavením překladače

(konec 2.části)

Analýza

- Celkem 13 nových prvků v jazyku
- Některé vyvolaly bouřlivé diskuze
- Žádná novinka není povinná, dosavadní kód bude i nadále fungovat

Analýza – generické typy

- Je to nejvýznamnější novinka
- Není to stejné jako šablony v C++
- Neříká se tomu „šablony“, ani „templaty“
- Vliv na knihovnu tříd BCL, zvláště kolekce
- Širší re-use, lepší typová kontrola

- Závěr: Je to velmi pozitivní věc

Analýza – nulovatelné typy

- Zjednodušení práce s SQL
- Teze: Je to půlkrok k regulárním výrazům
  int? = žádný nebo jeden int
  int! = právě jeden int
  int+ = jeden nebo více intů
  int* = libovolný počet intů
- Problémy:
  - Kolize int* s unsafe pointery
  - Stále nám chybí i opak = nenulovatelné typy

Analýza – zjednodušení práce

- Některé nové konstrukty zjednodušují práci:
  - Iterátory
  - Bezejmenné metody
  - Odlišná přístupová prává pro get a set
  - Kovariance a kontravariance

Analýza – další novinky

- Operátor :: zpřístupní global namespace
- Částečné deklarace zpřehlední kód
- Ostatní novinky jsou již méně důležité, spíše kosmetické

(konec 3.části)

Novinky ve vývojovém prostředí (stručný přehled)

1. Refactoring kódu
2. Expanze podle vzoru
3. Chytrá kontextová nápověda v debuggeru
4. Chytřejší sledování kontextu v IntelliSense
5. Možnost zobrazit metadata jako zdroják

…a mnoho dalšího…

1. Refactoring

- Rename
  - Přejmenuje prvek v deklaraci i volání
  - Bezpečnější, než klasické textové find & replace
- Extract method
  - Z části kódu udělá samostatnou metodu
- Encapsulate field
  - Datový prvek (proměnnou) zabalí do property
  - Změní kód používající proměnnou na property

Refactoring (pokračování)

- Extract interface
  - Vytvoří rozhraní podle existující třídy
- Promote local variable to parameter
  - Povýší lokální proměnou na parametr volání
- Remove parameters
  - Odstraní parametry z deklarace i volání
- Reorder parameters
  - Změní pořadí parametrů v deklaraci i volání

2. Expanze podle vzoru

- Je to „programování myší“ aneb „Insert code snippet“
- Tři možné aktivace:
  - Lokální menu (pravým tlačítkem)
  - Klávesová zkratka Ctrl+K,X
  - Zadat název snippetu a zmáčknout 2x Tab
- Příklad – foreach:

foreach(object var in collection_to_loop) { … }

Novinky v knihovně tříd BCL

- Base Class Library obsahuje také mnoho nových prvků (nad rámec této přednášky)