Zpravodaj moderníProgramování 06/2012: Parametry metod

Předpoklad ke studiu tohoto čísla: učebnice pro středně pokročilé (žlutá)

Parametry jsou upřesňující informace, dodávané zvenčí do metody z místa jejího volání. Významově 95 % výkladu o parametrech najdete v modré a především ve žluté učebnici. Na 4,5 % ze zbytku se podíváme v tomto čísle Zpravodaje.

Neurčitý počet parametrů

Příležitostně můžete chtít, aby metoda pracovala (víceméně stejným způsobem) pro jednu hodnotu, pro dvě hodnoty, pro tři hodnoty atd. Částečně je to možno řešit variantami metod (tzv. přetěžováním), ale to je přinejmenším pracné. Jazyk C# proto pomocí klíčového slova `params` umožňuje, aby **posledním** parametrem metody bylo buď pole, nebo předem neurčený počet hodnot stejného typu.

Pro ukázku si vytvořte ve *Visual C#* nový projekt. Volbou *Project* > *Add Class* do projektu vložte třídu `Pomůcky`, jejíž kód následuje:

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

namespace Neurčitý_počet_parametrů
{
   class Pomůcky
   {
      public static int Sečti(params int[] poleČísel)
      {
         int součet = 0;
         foreach (int číslo in poleČísel)
            součet += číslo;
         return součet;
      }
   }
}
```

Ve třídě je definována jedna statická metoda `Sečti`, která vypadá, jako že vypočte součet čísel předaných ji v poli jako parametr. Opravdu se takto dá použít, nicméně vzhledem k přítomnosti klíčového slova `params` lze také namísto pole čísel předat libovolné množství samostatných čísel, jak ukazuje kód `Load`u okna:

```
private void oknoProgramu_Load(object sender, EventArgs e)
{
  int[] pole = new int[] { 10, 20, 30 };
  MessageBox.Show(
    "Součet pole: " + Pomůcky.Sečti(pole).ToString() +
    Environment.NewLine +
    "Součet samostatných čísel: " +
                     Pomůcky.Sečti(1, 2, 3, 4, 5).ToString());
}
```

Pojmenované parametry

Postupem standardním na všech výpočetních platformách, jakým se předává více parametrů do metody, je předávání podle pořadí - první předaný parametr se ztotožní s prvním očekávaným, druhý s druhým atd. Pokud např. použijeme dvouparametrickou verzi metody `MessageBox.Show`, musí být prvním parametrem vlastní text zprávy a druhým parametrem text titulkové lišty okna se zprávou. Pořadí nelze zaměňovat.

Na konci r. 2007, jestli se nepletu, byla do jazyka C# zavedena možnost předávání parametrů podle jména. Pro ukázku vytvořte nový projekt a události `Load` a `Paint` obsluhujte podle následujícího kódu:

```
private void oknoProgramu_Load(object sender, EventArgs e)
{
   MessageBox.Show("Text zprávy obyčejně",
                   "Titulek zprávy obyčejně");
   MessageBox.Show(caption: "Titulek zprávy s pojm. par.",
                   text: "Text zprávy s pojm. par.");
}

private void oknoProgramu_Paint(object sender, PaintEventArgs e)
{
   Graphics kp = e.Graphics;
   kp.FillRectangle(Brushes.Red, 10, 10, 100, 100);
   kp.FillRectangle(brush: Brushes.Blue, x: 10, y: 120,
                    width: 100, height: 100);
}
```

Volání `MessageBox.Show` a `kp.FillRectangle` jsou uvedena nejprve obvyklým způsobem (předávání pořadím) a poté způsobem novým (předávání jmény). V případě volání `Show` je vidět, že pořadí parametrů může pak být libovolné.

Smyslem této inovace asi není to, abychom parametry předávali, jak se nám zlíbí - nač v tom dělat hokej, že? - ale mj. jistá možnost zpřehlednění kódu.[1] Do kódu totiž rovnou dokumentujete, co předáváte. Abych se přiznal, já to zatím nepoužívám, ale pro výukové účely to nemusí být úplně od věci, takže Vás s tím seznamuji.

V příkladu najdete také použití na vlastní metodě. Do projektu jsem přidal třídu `Pomůcky`:

```
class Pomůcky
{
   public static double Mocnina(double umocňovanéČíslo,
                                 double exponent)
   {
       return Math.Pow(umocňovanéČíslo, exponent);
   }
}
```

[1] Druhým důvodem jsou tzv. volitelné parametry, což je to zbývající půlprocento, viz úvod.

Její použití jsem napojil, pro příklad, na `MouseDown` okna:

```
private void oknoProgramu_MouseDown(object sender, MouseEventArgs e)
{
    double ŠestNaDruhou = Pomůcky.Mocnina(6, 2);
    double DvěNaŠestou = Pomůcky.Mocnina(umocňovanéČíslo: 2,
                                         exponent: 6);
    MessageBox.Show(
        "6^2 = " + ŠestNaDruhou.ToString() + Environment.NewLine +
        "2^6 = " + DvěNaŠestou.ToString());
}
```

Vstupně-výstupní parametry

Parametr je standardně vstupní informací do metody. Jestliže jej v metodě změníte, navenek se to neprojeví. Pro ukázku vložte do nového projektu jako obvykle třídu `Pomůcky`:

```
class Pomůcky
{
    public static void UpravČísloNefunguje(int číslo)
    {
        číslo++;
    }
}
```

V kódu okna obsluhujte nějaké tlačítko:

```
private void tlačítkoStandardníParametr_Click(object sender, EventArgs e)
{
    int číslo = 5;
    Pomůcky.UpravČísloNefunguje(číslo);
    MessageBox.Show(číslo.ToString());
}
```

Uživateli se zobrazí pětka, nikoli šestka. Jak o tom podiskutujeme za chvíli, v zásadě to tak má být. Dejme tomu, že bychom ale přeci jen hodnotu parametru i navenek změnit chtěli. Jazyk C# za tím účelem obsahuje klíčové slovo `ref`, kterým říkáme, že daný parametr slouží jak pro vstup, tak pro výstup informací z metody.

Rozšíříme třídu `Pomůcky` o další metodu:

```
public static void UpravČísloFunguje(ref int číslo)
{
    číslo++;
}
```

kterou zavoláme na nějaké dalším tlačítku okna:

```
private void tlačítkoRefParametr_Click(object sender, EventArgs e)
{
    int číslo = 5;
    Pomůcky.UpravČísloFunguje(ref číslo);
    MessageBox.Show(číslo.ToString());
}
```

Program nyní zobrazí šestku. Změny parametru se promítnou z metody navenek. Všimněte si, že klíčové slovo `ref` musí být použito jak v definici metody, tak v místě jejího volání.

Vstupně-výstupní parametry jsou trochu kontroverzní záležitostí. Řada autorů proti nim brojí, považují je za postranní efekt metody, který program znepřehledňuje. Zřejmě z tohoto důvodu možnost vstupně-výstupních parametrů neexistuje např. v Javě, starším to bratru .NET/C#. V C# se na možný postranní efekt alespoň upozorňuje povinným slovem `ref` v místě volání.

Jak by se naše úloha vyřešila způsobem, proti kterému nikdo brojit nebude, ukazují následující výpisy:

```
public static int UpravČísloLepší(int číslo)
{
   return číslo + 1;
}
```

```
private void tlačítkoLepšíŘešení_Click(object sender, EventArgs e)
{
   int číslo = 5;
   int upravenéČíslo = Pomůcky.UpravČísloLepší(číslo);
   MessageBox.Show(upravenéČíslo.ToString());
}
```

Na závěr poznámka k terminologii, se kterou se můžete setkat. Klasický způsob předávání parametrů, tj. pouze jako vstupy metody, se nazývá ***volání hodnotou***. Do metody se překopíruje hodnota venkovní proměnné. Naproti tomu u vstupně-výstupních parametrů mluvíme o tzv. ***volání odkazem*** - do metody se překopíruje odkaz na venkovní proměnnou, a proto může metoda hodnotu venkovní proměnné změnit.

Parametry a odkazové typy

Dosavadní diskuse vstupně-výstupních parametrů se týkala především hodnotových typů, což jsou číselné typy a dále `char`, `bool` a všechny zjednodušené třídy `struct`. Pokud jde o odkazové typy (pole, seznamy, všechny třídy `class`), tam VŽDY pracujeme pouze s odkazem (na pole, na objekt, ...), což znamená, že metoda může např. změnit hodnoty složek pole, aniž se používá slovo `ref` (teď nevím, jak se k tomu staví teoretici, opět jde o postranní efekt... Jde to i Javě...):

```
public static void ZměnaPoložek(int[] poleČísel)
{
   for (int index = 0; index < poleČísel.Length; index++)
       poleČísel[index]++;
}
```

```
private void tlačítkoZměnaPoložek_Click(object sender, EventArgs e)
{
   int[] pole = new int[] { 10, 20, 30 };
   Pomůcky.ZměnaPoložek(pole);
   MessageBox.Show(PoleProVýpis(pole));
}
```

Vypíše se 11, 21, 31. Používáme pomocnou metodu:

```
// Pomocná metoda
private string PoleProVýpis(int[] poleČísel)
{
    string zpráva = "";
    foreach (int číslo in poleČísel)
        zpráva += číslo.ToString() + Environment.NewLine;
    return zpráva;
}
```

Klíčové slovo `ref` uplatníme v souvislosti s poli (a jinými odkazovými typy), pokud má metoda **změnit celé pole**, např. změnit počet jeho prvků. Ukažme si to příkladu, kdy chceme pole upravit případným přidáním prvku tak, aby jeho součet byl aspoň 100:

```
public static void ZměnaCelého(ref int[] poleČísel)
{
    // Sečte čísla z pole
    int součet = 0;
    foreach (int číslo in poleČísel)
        součet += číslo;

    if (součet < 100)
    {
        // Vytvoří nové, o položku delší pole
        int novýPočet = poleČísel.Length + 1;
        int[] novéPole = new int[novýPočet];

        // Rychle zkopíruje dosavadní obsah
        Array.Copy(poleČísel, novéPole, poleČísel.Length);

        // Naplní dodatečnou položku a UPRAVÍ ODKAZ
        novéPole[novýPočet - 1] = 100 - součet;
        poleČísel = novéPole;
    }
}
```

```
private void tlačítkoZměnaCelého_Click(object sender, EventArgs e)
{
    int[] pole = new int[] { 10, 20, 30 };
    Pomůcky.ZměnaCelého(ref pole);
    MessageBox.Show(PoleProVýpis(pole));
}
```

Samozřejmě, jde to i bez `ref`, stejně jako u příkladu s číslem:

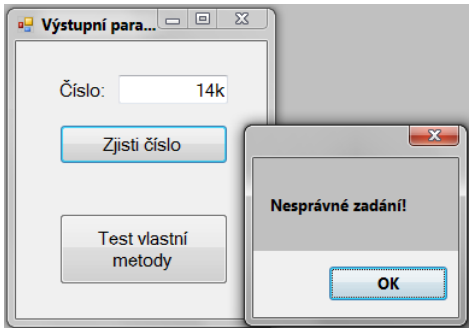
```
public static int[] LepšíZměnaCelého(int[] poleČísel)
{
    // Sečte čísla z pole
    int součet = 0;
    foreach (int číslo in poleČísel)
        součet += číslo;

    if (součet < 100)
    {
        // Vytvoříé nové, o položku delší pole
        int novýPočet = poleČísel.Length + 1;
        int[] novéPole = new int[novýPočet];

        // Rychle zkopíruje dosavadní obsah
        Array.Copy(poleČísel, novéPole, poleČísel.Length);

        // Naplní dodatečnou položku a VRÁTÍ VÝSLEDEK
        novéPole[novýPočet - 1] = 100 - součet;
        return novéPole;
    }
    else // VRÁTÍ PŮVODNÍ, NEZMĚNĚNÉ POLE
        return poleČísel;
}
```

```
private void tlačítkoLepšíZměnaCelého_Click(object sender, EventArgs e)
{
    int[] pole = new int[] { 10, 20, 30 };
    pole = Pomůcky.LepšíZměnaCelého(pole);
    MessageBox.Show(PoleProVýpis(pole));
}
```

Výstupní parametry

Dnešní výklad zakončím seznámením s parametry, které slouží pouze pro výstup hodnot z metody. Jelikož základním výstupem metody je její návratová hodnota, mohou se výstupní parametry uplatnit asi jen v případech, kdy potřebujeme ven z metody dostat hodnot vícero.

Příkladem použití z knihoven .NETu je metoda `int.TryParse`, která plní podobnou funkci jako volání `Convert.ToInt32` zabalené do `try-catch`. Metoda vrací jednak číslo vyrobené z řetězce a jednak, zda se převod vůbec povedl či nikoli. Vyzkoušet si ji můžete na novém projektu:



```
private void tlačítkoZjistiČíslo_Click(object sender, EventArgs e)
{
    int číslo;
    bool ok = int.TryParse(poleČíslo.Text, out číslo);
    if (ok)
        MessageBox.Show("Zadáno číslo " + číslo.ToString());
    else
        MessageBox.Show("Nesprávné zadání!");
}
```

Program ukazuje ještě použití výstupních parametrů na vlastní metodě `Počty`, která počítá součet a součin čísel předaných jí jako parametry. Metoda v přidané třídě `Pomůcky`:

```
class Pomůcky
{
    public static void Počty(int číslo1, int číslo2,
                             out int součet, out int součin)
    {
        součet = číslo1 + číslo2;
        součin = číslo1 * číslo2;
    }
}
```

a její volání z okna:

```
private void tlačítkoTest_Click(object sender, EventArgs e)
{
    int výsledekSoučet, výsledekSoučin;
    Pomůcky.Počty(2, 10, out výsledekSoučet, out výsledekSoučin);
    MessageBox.Show(
      "Součet: " + výsledekSoučet.ToString() + Environment.NewLine +
      "Součin: " + výsledekSoučin.ToString());
}
```

Stejně jako u `ref` i zde si všimněte, že slovo `out` je třeba zapsat jak v definici metody, tak v místě jejího volání.

Stejně jako u `ref` i zde jde o kontroverzní záležitost. Např. v Javě `out` není. Jak naši úlohu vyřešit lepším způsobem, bez použití `out`, tedy s tím, že výstupem metody bude pouze její návratová hodnota, ukazuje Úloha 9-J-04 (Přepravka) z oranžové sbírky.

Radek Vystavěl, 4. června 2012

Pokud Vám Zpravodaje moderníProgramování připadají užitečné, doporučte jejich odběr svým známým. Mohou se přihlásit na webu www.moderniProgramovani.cz.