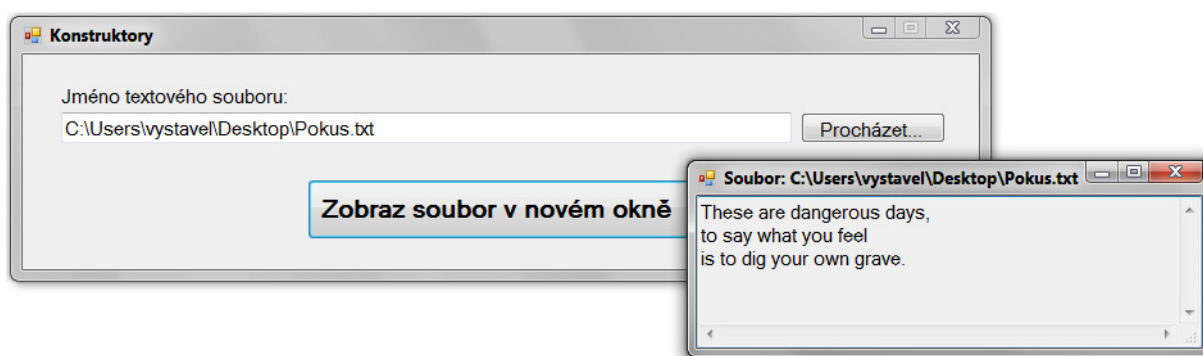


# Zpravodaj moderní Programování 01/2012: Vytváření objektů - konstruktory a tovární metody

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

V tomto čísle Zpravodaje se budeme věnovat vytváření objektů, prostudujeme jak konstruktory, tak tovární metody. Prakticky si tuto problematiku ilustrujeme na programu, který v hlavním okně od uživatele zjistí jméno souboru a následně tento soubor ve druhém okně otevře. Právě druhé okno bude objektem, jehož vytvářením se budeme zabývat (a vším, co s tím souvisí).



Vytváření dalších oken pomocí *Project > Add Windows Form* znáte z 11. kapitoly žluté učebnice. V našem případě potřebujeme z hlavního do druhého okna předat informaci o jméně souboru. Pro tyto účely může druhé okno definovat veřejnou metodu `NastavJménoSouboru` nebo veřejnou vlastnost `JménoSouboru`. Hlavní okno pak metodu zavolá, resp. vlastnost nastaví tak, jak popisuje zmíněná 11. kapitola.

## Konstruktor

Informace o jméně souboru je však pro náš studovaný objekt, druhé okno, zcela zásadní, bez ní vlastně nemůže pracovat. Uvážíme-li lidskou zapomnětlivost, možná by bylo lepší nespoléhat na to, že po vytvoření druhého okna standardně vygenerovaným bezparametrickým konstruktorem zavoláme zmíněnou metodu, resp. nastavíme zmíněnou vlastnost. Možná by bylo lepší zadání této informace vynutit již při vytváření okna. To nás již vede k definici jednoparametrického konstruktora, který požadovanou informaci převezme ve svém parametru.

Tento postup najdete v knihovnách .NET. Vzpomeňte například na

- `new Pen(Color.CornflowerBlue)` ze 7. kapitoly modré učebnice,
- nebo `new StreamReader("pokus.txt")` z 5. kapitoly žluté učebnice.

Nemá moc smyslu vytvářet pero bez uvedení barvy, či otevírat soubor bez uvedení jména.

Vytvořte nový projekt ze šablony „Windows Forms Application“, uživatelské rozhraní hlavního okna upravte dle výše uvedeného obrázku a standardním způsobem naprogramujte tlačítko „Procházet...“. Do projektu přidejte druhé okno nazvané např. `OknoZobrazeníSouboru`. Do něj vložte textové pole `Zobrazení` a nastavte tyto jeho vlastnosti:

- MultiLine a ReadOnly na True;
- TabStop a WordWrap na False;
- ScrollBars na Both;
- Dock na Fill.

Kód druhého okna pak upravte následovně:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Nutno přidat odkaz na jmenný prostor, kde se nachází třída File
using System.IO;

namespace Konstruktory
{
    public partial class OknoZobrazeníSouboru : Form
    {
        // Okno si jméno souboru uloží do této proměnné
        // (pro budoucí potřeby)
        private string jménoSouboru;

        public OknoZobrazeníSouboru()
        {
            InitializeComponent();
        }

        public OknoZobrazeníSouboru(string jménoSouboru)
            : this() // Spustíme nejprve bezparametrický konstruktor
        {
            // Uložíme do vlastní proměnné
            this.jménoSouboru = jménoSouboru;

            // Text titulní lišty upravíme podle jména souboru
            Text = "Soubor: " + jménoSouboru;

            // Zobrazíme obsah souboru
            try
            {
                poleZobrazení.Text = File.ReadAllText
                    (jménoSouboru, Encoding.Default);
            }
            catch
            {
                poleZobrazení.Text = "Soubor nelze načíst...";
            }
        }
    }
}
```

Konstruktor je metoda, která se zavolá při vytváření nové instance dané třídy. Co náš konstruktor dělá? Co je potřeba provést při vytváření nového okna pro zobrazení souboru? Čtème kód odspodu:

- Již v okamžiku vytvoření okna textové pole naplníme obsahem souboru. Využíváme metodu `File.ReadAllText`, která celý textový soubor najednou načte do jednoho textového řetězce. Jindy se zase hodí `File.ReadAllLines`, která vytváří pole řádků.
- Dle jména souboru upravíme text titulkové lišty okna.
- Hodnotu parametru si pro případné budoucí použití uložíme do soukromé členské proměnné okna. Pokud bude chtít druhé okno s touto informací dále pracovat, musí si ji někde pamatovat. Parametr konstruktoru po vytvoření objektu přestane existovat. V našem jednoduchém případě k žádnému budoucímu použití nedojde, nicméně chtěl jsem ukázat typickou situaci.  
Všimněte si, že parametr se jmenuje stejně jako členská proměnná. Díky tomuto konfliktu jmen parametr v kódu konstruktoru zastíní členskou proměnnou. Abychom se na ni dostali, její název blíže určíme slovem `this`, odkazem objektu na sebe samého.
- Má-li objekt více konstruktorů, obvykle spolu souvisejí a mnohdy se navzájem volají (aby se nějaký kus kódu neduplikoval). Náš jednoparametrický konstruktor potřebuje před svým spuštěním provést kód bezparametrického konstruktoru vygenerovaného ze šablony, kde dochází k volání Designerova kódu. Za tím účelem je pomocí dvojtečky a slova `this` v hlavičce našeho konstruktoru volán bezparametrický konstruktor téže třídy.

V hlavním okně po stisku hlavního tlačítka konstruktor zavoláme:

```
private void tlačítkoZobrazSoubor_Click(object sender, EventArgs e)
{
    OknoZobrazeníSouboru okno =
        new OknoZobrazeníSouboru(poleJménoSouboru.Text);
    okno.Show();
}
```

### Skrytí bezparametrického konstruktoru

Při volání konstruktoru druhého okna se stále nabízí i bezparametrická varianta. Tu bychom ale chtěli potlačit, aby opravdu nebylo zbylí a okno bez zadání jména souboru nešlo vytvořit. Možným řešením je bezparametrický konstruktor označit jako soukromý:

```
// Změníme modifikátor přístupu bezparametrického konstruktoru
private OknoZobrazeníSouboru()
{
    InitializeComponent();
}
```

Ověřte, že nyní to bez zadání parametru už určitě nepůjde.

Samozřejmě, bylo by možné jej také úplně zrušit, volání `InitializeComponent` přesunout do našeho konstrukturu a ještě odstranit `:this()`. Ponechávám zde však raději řešení s modifikátorem `private`, jelikož některé nástroje vývojového prostředí špatně nesou, když objekt nemá bezparametrický konstruktore (např. plocha Designeru se někdy může zcela rozpadnout).

## Tovární metoda

Některé objekty mohou být navrženy tak, že volání konstrukturu je zabaleno do tzv. tovární metody. Objekt pak nevytváříme pomocí `new`, ale právě voláním této metody. Příkladem z knihoven .NET může být míchání barvy pomocí `Color.FromArgb`, které vytvoří novou instanci `Color`, aniž explicitně píšeme `new`. Možné důvody pro vytváření továrních metod:

- Chceme jméno, které výstižně označuje způsob vytváření objektu (to u konstrukturu nejde, ten se v C# vždy jmenuje stejně jako třída).
- Chceme kontrolovat, že se vytvoří pouze jediná instance dané třídy.

V našem příkladu nejprve skryjeme nedávno vytvořený konstruktore:

```
// I tento konstruktore navenek skryjeme
private OknoZobrazeníSouboru(string jménoSouboru)
    : this()
{
    ...
}
```

Poté ve třídě druhého okna doplníme tovární metodu:

```
// Tovární metoda
public static OknoZobrazeníSouboru NovéOkno(string jménoSouboru)
{
    OknoZobrazeníSouboru okno = new OknoZobrazeníSouboru(jménoSouboru);
    return okno;
}
```

Všimněte si, tovární metoda musí být statická. Nemůže to být běžná metoda, tj. metoda instance. Nechceme vytvářet instanci, abychom mohli vytvořit instanci... Na webu MSDN si můžete zkontrolovat, že `Color.FromArgb` je rovněž statickou metodou.

Z kódu hlavního okna potom druhé okno vytvoříme a zobrazíme takto:

```
private void tlačítkoZobrazSoubor_Click(object sender, EventArgs e)
{
    OknoZobrazeníSouboru.NovéOkno(poleJménoSouboru.Text).Show();
}
```

*Radek Vystavěl, 2. ledna 2012*

*Pokud Vám Zpravodaje moderníProgramování připadají užitečné, doporučte jejich odběr svým známým. Mohou se přihlásit na webu [www.moderniProgramovani.cz](http://www.moderniProgramovani.cz).*