Kapitola 4. Přiřazovací příkaz

V předchozí kapitole jsme si říkali, že život do programů s grafickým uživatelským rozhraním vnášejí události a že ke každé události, na kterou má program reagovat, je třeba připojit obslužnou metodu. Do obslužných metod se zapisují příkazy, jenž má počítač v reakci na vznik události provést. V našich příkladech obslužných metod jsme zatím zapisovali příkazy volání předem připravených podprogramů, konkrétně `Close` a `MessageBox.Show`. V této kapitole se zaměříme na jiný druh příkazu, konkrétně na přiřazovací příkaz. Ten umožňuje data kopírovat z jednoho místa na jiné ať už beze změny, nebo nějak vhodně zkombinovaná.

KOPÍROVÁNÍ TEXTOVÉHO POLE

Použití přiřazovacího příkazu si opět nejlépe ukážeme na příkladech. Začneme prográmkem, jehož uživatelské rozhraní je zobrazeno na Obr. 26. Budou v něm dvě textová pole doprovázená popisky a jedno tlačítko. Ve spodním textovém poli bude zakázána editace, bude určeno „jen ke čtení“. Funkce programu bude spočívat v tom, že při stisku tlačítka „Kopíruj“ se zkopíruje text zadaný uživatelem v horním poli do pole spodního.

Obr. 26 Úvodní prográmek s přiřazovacím příkazem

Ve *Visual Studiu* vytvořte nový projekt nazvaný „Kopírování“ a v Návrháři připravte uživatelské rozhraní podle obrázku. Horní textové pole pojmenujte `poleVěta`, spodní `poleKopie` a tlačítko `tlačítkoKopíruj`.

Jelikož se akce, tj. kopírování, má spustit v reakci na stisk tlačítka, je potřeba vložit obslužnou metodu události `Click` tlačítka. Jelikož `Click` je hlavní událostí tlačítka, stačí na tlačítko poklepat přímo v Návrháři. Na místě, kde bude blikat kurzor, vložte tento příkaz (opět jej opisujte pečlivě včetně velkých a malých písmen):

```
    poleKopie.Text = poleVěta.Text;
```

Celou obslužnou metodu pak zachycuje Výpis 5. Program spusťte a ověřte jeho funkčnost.

```
private void tlačítkoKopíruj_Click(object sender, EventArgs e)
{
    poleKopie.Text = poleVěta.Text;
}
```

Výpis 5 Obslužná metoda události `Click` tlačítka

PŘIŘAZOVACÍ PŘÍKAZ

Jak to funguje? Co ten příkaz znamená? Příkaz, který jste do programu vložili, je tzv. ***přiřazovací příkaz***, který **hodnotu na pravé straně od rovnítka přiřadí na místo udané na straně levé**. Obecně bychom jej mohli vyjádřit takto (tento zápis si velmi dobře zapamatujte, budete jej od nynějška používat neustále):

```
kam = co;
```

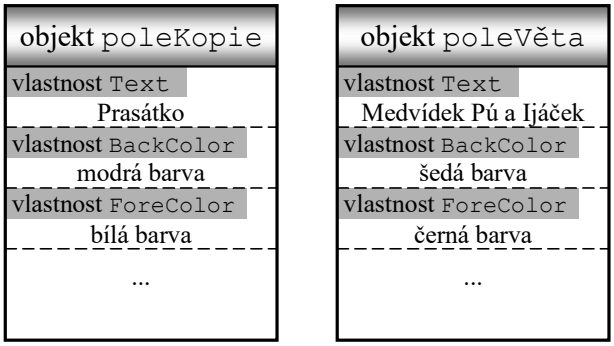
Porovnáním obecného tvaru s příkazem zapsaným do programu (Tab. 12) můžete vidět, že zapsaný příkaz přiřazuje hodnotu vlastnosti `Text` objektu `poleVěta` do vlastnosti `Text` objektu `poleKopie`. Jelikož vlastnost `Text` reprezentuje text v textovém poli zobrazený, bude po provedení výše uvedeného příkazu `poleKopie` zobrazovat tentýž text jako `poleVěta`. Pokud uživatel následně obsah `poleVěta` změní, nebudou se již dále zobrazované texty shodovat. K jejich synchronizaci dojde opět při dalším stisku tlačítka, který vyvolá vstup do obslužné metody a provedení přiřazovacího příkazu.

Obecně	*V zapsaném příkazu*	*Jak to počítač přečte*
kam	`poleKopie.Text`	Do vlastnosti `Text` objektu `poleKopie` …
=	=	… přiřaď …
co	`poleVěta.Text`	… hodnotu vlastnosti `Text` objektu `poleVěta`.

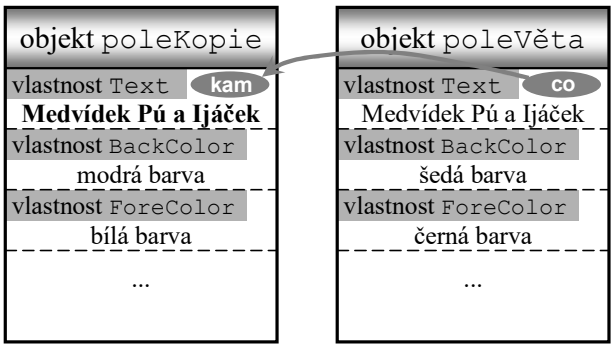
Tab. 12 Porovnání obecného zápisu s použitým příkazem

Z uvedeného výkladu jste se zřejmě dovtípili, že se v zapsaném příkazu provádí **bližší určování jménem objektu**. Je použito schéma „jméno objektu, tečka, jméno vlastnosti daného objektu“. Vlastnost `Text` mají všechny objekty uživatelského rozhraní, proto je vždy potřeba říct, vlastnost `Text` **kterého objektu** máte na mysli. Zápis `poleVěta.Text` tedy znamená vlastnost `Text` objektu `poleVěta`, podobně zápis `poleKopie.Text` odkazuje na vlastnost `Text` objektu `poleKopie`.

Činnost přiřazovacího příkazu ve spojení s objekty si můžete představit i podle schématu z Obr. 27 a Obr. 28, které zachycují situaci před provedením přiřazení a bezprostředně po něm. Objekty `poleKopie` i `poleVěta` jsou na něm zobrazeny jako určité kontejnery obsahující řadu přihrádek. Každá přihrádka je jedna vlastnost objektu, hodnota vlastnosti je obsah přihrádky. Provedením výše uvedeného přiřazení se vezme obsah přihrádky `Text` objektu `poleVěta` a tak, jak je, se **zkopíruje** do přihrádky `Text` objektu `poleKopie`, kde nahradí jakýkoli původní obsah. Obsah přihrádky, ze které se kopírovalo, zůstane nedotčen. Stejně tak zůstane nedotčen obsah všech ostatních přihrádek obou objektů.



Obr. 27 Objekty před přiřazením



Obr. 28 Objekty po realizaci přiřazení

DALŠÍ PŘIŘAZOVÁNÍ

V úvodním příkladu kapitoly jsme použili přiřazovací příkaz, který obsah vlastnosti jednoho objektu kopíroval do vlastnosti jiného objektu. Nyní si ukážeme, že do přihrádky-vlastnosti lze přiřadit nejen hodnotu z jiného objektu, ale i hodnotu, která je ve zdrojovém textu programu přímo uvedená.

Obr. 29 Program s přiřazováním přímo uvedených hodnot

Program, na jehož příkladu si přiřazování přímo uvedených hodnot ukážeme, bude mít podobu podle Obr. 29. Textové pole umožní uživateli zadat slovo či větu a navíc ještě bude ovládáno jednotlivými tlačítky:

- Tlačítko „Ahoj lidi“ vloží do textového pole takto znějící větu.
- Tlačítko „Vymaž“ obsah textového pole smaže.
- Tlačítko „Text červeně“ obarví text v poli na červeno.
- Tlačítko „Zablokuj“ textové pole zablokuje tak, že text v něm nebude možno editovat ani přenášet do schránky.
- Tlačítko „Odblokuj“ vrátí akci tlačítka „Zablokuj“.

Ve vývojovém prostředí *Visual Studio* vytvořte nový projekt a podle Obr. 29 připravte uživatelské rozhraní. Abychom se v dalším postupu spolu shodovali, použijte tato pojmenování jednotlivých objektů: `oknoProgramu`, `popisekVěta`, `poleVěta`, `tlačítkoAhojLidi`, `tlačítkoVymaž`, `tlačítkoTextČerveně`, `tlačítkoZablokuj` a `tlačítkoOdblokuj`.

Nyní se můžete pustit do programování naplánovaných činností. Jelikož u všech tlačítek půjde o obsluhu hlavní události `Click`, nebude ani potřeba použít Okno vlastností. Proveďte následující kroky:

- V Návrháři poklepejte na tlačítko „Ahoj lidi“ a do vložené obslužné metody `tlačítkoAhojLidi_Click` vložte příkaz

```
poleVěta.Text = "Ahoj lidi";
```

  Do vlastnosti `Text` textového pole se tím přiřadí přímo uvedený řetězec. Ten je z obou stran ohraničen uvozovkami a ve vývojovém prostředí zobrazen červeně. S tím jsme se už setkali při předávání parametrů do podprogramu `MessageBox.Show`.

- V Návrháři poklepejte na tlačítko „Vymaž“ a do vložené obslužné metody `tlačítkoVymaž_Click` vložte příkaz

```
poleVěta.Text = "";
```

  nebo příkaz

```
poleVěta.Text = null;
```

  První varianta pracuje s tzv. ***prázdným řetězcem*** zapsaným dvojicí uvozovek bezprostředně za sebou bez jakékoli mezery mezi. Ve variantě alternativní představuje klíčové slovo `null` tzv. ***nedefinovanou hodnotu***, používanou i jinde než pouze při práci s řetězci. Prázdný řetězec a nedefinovaná hodnota není obecně jedno a totéž, ve většině případů však namísto prázdného řetězce hodnotu `null` zapsat můžete.

- V Návrháři poklepejte na tlačítko „Text červeně“ a do vložené obslužné metody `tlačítkoTextČerveně_Click` vložte příkaz

```
poleVěta.ForeColor = Color.Red;
```

Pracujeme stále s objektem `poleVěta`, nyní však s jeho jinou vlastností. Konkrétně s vlastností `ForeColor` udávající barvu zobrazeného textu. Konkrétní hodnota barvy se zapíše slovem `Color`, za kterým následuje tečka a anglický název barvy.

V momentě, kdy za slovem `Color` napíšete tečku, vám vývojové prostředí okamžitě všechny použitelné názvy barev nabídne. Možná jste si tohoto ***našeptávání*** ze strany vývojového prostředí již všimli – ať cokoli píšete, *Visual Studio* se vám snaží napovědět možné pokračování, případně za vás slovo dokončí po klepnutí na variantu či stisku klávesy *Tab*. Našeptávání je výborná věc, zvykněte si je používat! Ušetříte si tak spoustu překlepů.

- V Návrháři poklepejte na tlačítko „Zablokuj“ a do vložené obslužné metody `tlačítkoZablokuj_Click` vložte příkaz

```
poleVěta.Enabled = false;
```

Tento příkaz pracuje s vlastností `Enabled` textového pole. Je-li tato vlastnost nastavená na `true`, pole normálně pracuje, je-li nastavená na `false`, pole dělá „mrtvého brouka“.

Všimněte si, že hodnoty **`true` a `false` se ve zdrojovém textu zapisují vždy malými písmeny**! To je rozdíl oproti Oknu vlastností, který tyto hodnoty píše s prvním písmenem velkým. Tvůrci *Visual Studio* se pro tuto nekonzistenci rozhodli, nám zůstává toto rozhodnutí respektovat.

- No a konečně poklepejte na tlačítko „Odblokuj“ a do vložené obslužné metody `tlačítkoOdblokuj_Click` vložte příkaz

```
poleVěta.Enabled = true;
```

Vysvětlení je stejné jako u tlačítka „Zablokuj“.

Pro kontrolu, jak by měl nakonec vypadat celý zdrojový text `Form1.cs`, se podívejte na Výpis 6. Pro jistotu bych zdůraznil, že zdrojový text `Form1.cs` je jen jednou částí příběhu. Není možné tedy přípravu programu „vyřešit“ tak, že tento zdrojový text od začátku do konce opíšete! Bez výše popisované práce s Návrhářem se neobejdete. Návrhář vytvoří vazby mezi událostmi a jejich obsluhami a tyto vazby zapíše do jím obhospodařovaného kódu `Form1.Designer.cs`.

Program spusťte a důkladně prověřte všechny jeho funkce. Pokud by se vám jeho kompletace nepodařila, najdete ho jako obvykle na webových stránkách učebnice.

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

namespace Různá_přiřazení
{
  public partial class oknoProgramu : Form
  {
    public oknoProgramu()
    {
        InitializeComponent();
    }

    private void tlačítkoAhojLidi_Click(object sender,
                                        EventArgs e)
    {
        poleVěta.Text = "Ahoj lidi";
    }

    private void tlačítkoVymaž_Click(object sender,
                                     EventArgs e)
    {
        poleVěta.Text = null;
    }

    private void tlačítkoTextČerveně_Click(object sender,
                                           EventArgs e)
    {
        poleVěta.ForeColor = Color.Red;
    }

    private void tlačítkoZablokuj_Click(object sender,
                                        EventArgs e)
    {
        poleVěta.Enabled = false;
    }

    private void tlačítkoOdblokuj_Click(object sender,
                                        EventArgs e)
    {
        poleVěta.Enabled = true;
    }
  }
}
```

Výpis 6 Zdrojový text `Form1.cs` programu *Různá přiřazení*

VÍCE PŘÍKAZŮ V METODĚ

Pohled na naše dosavadní programy *Události tlačítka*, *Kopírování* a *Různá přiřazení* by někoho mohl svádět k představě, že v obslužných metodách bývá vždy jediný příkaz, že se v nich vždy provede jediná akce. Není tomu tak, dovnitř metody můžete vložit tolik příkazů, kolik potřebujete. Hned se na nějaký program s více příkazy v metodě podíváme. Vraťte se k Obr. 29 a uvažujte se mnou.

Což takhle pozměnit funkci tlačítek „Zablokuj“ a „Odblokuj“? Možná by bylo šikovné, kdyby bylo „v provozu“ jen to, které má v daném stavu programu svůj smysl. Když například textové pole normálně pracuje, nemá význam mačkat tlačítko „Odblokuj“. Tlačítko by pouze nabízelo zbytečnou možnost odblokovat nezablokované pole. Podobně můžeme uvažovat i v záležitosti tlačítka „Zablokuj“.

Proveďme to tedy tak, že v obsluze tlačítka „Odblokuj:

- jednak odblokujeme textové pole,
- ale přitom současně vyřadíme (zablokujeme) tlačítko „Odblokuj“
- a navíc oživíme tlačítko „Zablokuj“.

Obsluha tlačítka „Zablokuj“ bude analogická:

- zablokujeme textové pole,
- přitom vyřadíme tlačítko „Zablokuj“
- a oživíme tlačítko „Odblokuj“.

Obslužné metody obou tlačítek se tedy změní do podoby, kterou ukazuje Výpis 7. Zkuste podle něj upravit program ve vašem počítači a spustit ho.

```
private void tlačítkoZablokuj_Click(object sender, EventArgs e)
{
   poleVěta.Enabled = false;
   tlačítkoZablokuj.Enabled = false;
   tlačítkoOdblokuj.Enabled = true;
}

private void tlačítkoOdblokuj_Click(object sender, EventArgs e)
{
   poleVěta.Enabled = true;
   tlačítkoOdblokuj.Enabled = false;
   tlačítkoZablokuj.Enabled = true;
}
```

Výpis 7 Upravené obsluhy tlačítek „Zablokuj“ a „Odblokuj“

Při testování programu jste asi zaznamenali jeden jeho nedostatek, a tím je skutečnost, že při startu jsou aktivní obě tlačítka. Nuže, není nic snazšího, než s pomocí Okna vlastností nastavit vlastnost `Enabled` tlačítka „Odblokuj“ na počáteční hodnotu `False`. Kompletní program najdete jako obvykle na webových stránkách učebnice.

K otázce více příkazů v metodě ještě zmíním, že většinou se každý příkaz z důvodu přehlednosti zapisuje na samostatný řádek zdrojového kódu. Není to však absolutní nutností, jazyk C# to nevyžaduje. Sem tam může být vhodnější na jeden řádek zapsat příkazů více, anebo naopak nějaký delší příkaz rozdělit na několik řádků[1]. Rozhodujícím kritériem je vždy přehlednost programu a to, aby jeho formální podoba co nejlépe odrážela logiku řešení.

SEKVENČNÍ ZPRACOVÁNÍ

Věnujme se i nadále problematice více akcí v jedné metodě. Zkusíme vytvořit program, který bude udržovat dvě zálohy dokumentu. No dokumentu, to je silné slovo, spíš bych měl říct jedné věty. V reakci na stisk tlačítka „Zálohuj“ se text z pole „Věta“ zkopíruje (zazálohuje) do pole „Záloha“. Současně s tím se záloha samotná také zazálohuje do třetího textového pole (Obr. 30).

Nuže, vytvořte si nový projekt „Dvojitá záloha“ a připravte uživatelské rozhraní podle Obr. 30. Abychom se shodovali, zvolte pojmenování textových polí stejné jako já: `poleVěta`, `poleZáloha`, `polePůvodníZáloha`.

Obr. 30 Program ilustrující otázku sekvenčního zpracování

V programu bude evidentně jediná obslužná metoda, a tou je obsluha události `Click` tlačítka. Vytvořte ji a zapište do ní jednak příkaz pro zkopírování věty do zálohy, jednak pro uložení původní zálohy. Kopírování z jednoho textového pole do druhého jsme dělali na začátku kapitoly, takže by neměl být problém obslužnou metodu zapsat, viz Výpis 8.

```
private void tlačítkoZálohuj_Click(object sender, EventArgs e)
{
   poleZáloha.Text = poleVěta.Text;
   polePůvodníZáloha.Text = poleZáloha.Text;
}
```

Výpis 8 Obsluha stisku tlačítka v programu *Dvojitá záloha*

Program spusťte a zkontrolujte jeho činnost. Dost možná se budete divit. Zapsat obě akce do jedné metody problém opravdu nebyl, akorát že program nefunguje, jak má. V obou zálohovacích textových polích je vždy stejná hodnota. Program tedy dvě zálohy nevytváří!

Příčina problému tkví v tom, že počítač nevykonává příkazy najednou. **Jednotlivé příkazy metody počítač provádí vždy sekvenčně, jeden po druhém**, v pořadí

[1] Rozdělení příkazu je možno provést pouze v tom místě, kde se dá zapsat mezera. To znamená buď tam, kde mezera už je, nebo např. před závorkou či za závorkou. Rozdělení není možno provést uprostřed přímo uvedeného řetězce uzavřeného v uvozovkách.

shora dolů, jak jsou zapsány ve zdrojovém kódu. V kódu se tedy nejprve obsah `poleVěta` zkopíruje do `poleZáloha` a až **poté** se `poleZáloha` zkopíruje do `polePůvodníZáloha`. V té chvíli je však již původní obsah `poleZáloha` nenávratně ztracen, přepsán obsahem `poleVěta`. Program tedy v obou „zálohových" polích vždy zobrazí tutéž hodnotu.

Řešením problému je, jak se asi rychle dovtípíte, prohodit příkazy mezi sebou. Nejprve je třeba do spodního pole zkopírovat dosavadní obsah `poleZáloha` a teprve poté, až je záloha bezpečně zazálohována, zkopírovat obsah `poleVěta` do `poleZáloha`. Správné řešení ukazuje Výpis 9. Kompletní program najdete opět na webu.

```
private void tlačítkoZálohuj_Click(object sender, EventArgs e)
{
   polePůvodníZáloha.Text = poleZáloha.Text;
   poleZáloha.Text = poleVěta.Text;
}
```

Výpis 9 Obsluha stisku tlačítka v programu *Dvojitá záloha* (už správná!)

SÉMANTICKÉ CHYBY

Možná se ptáte, jak je možné, že chybné pořadí příkazů neodhalilo vývojové prostředí podobně jako třeba zapomenutý středník. Neodhalilo ji proto, že ji ani nemohlo odhalit. Vývojové prostředí dokáže zkontrolovat, zda je program zapsán podle pravidel jazyka C#, nemůže však vědět, jestli jste uvedené pořadí příkazů náhodou nezvolili záměrně. Co když jste to tak opravdu chtěli? Jednu věc vývojové prostředí rozhodně nedokáže, a to vidět programátorům do hlavy.

Uvedenému druhu chyb, kdy je program funkční, ale dělá něco jiného, než dělat má, se říká ***chyby sémantické*** neboli významové. Chybám, se kterými jsme se setkali dříve, jako byl třeba zapomenutý středník, se oproti tomu říká ***chyby syntaktické***. To jsou prohřešky proti ***syntaxi*** daného programovacího jazyka.

Z počátku budete zápasit asi především se syntaktickými chybami. Jak si ale na pravidla jazyka C# víc a víc přivyknete, zjistíte, že skutečným problémem jsou chyby sémantické. Syntaktické chyby odhaluje vývojové prostředí za vás, chyby sémantické musíte najít sami. Jejich zrádnost je navíc ještě v tom, že mnohdy nejsou hned vidět, že se třeba projevují jen někdy.

Z důvodu převážně skrytého charakteru sémantických chyb se při profesionálním vývoji programů věnuje nemalý čas na testování programu. Tester, což je obvykle osoba odlišná od autora programu, se snaží v programu navodit co možná nejrůznější situace, zkouší používat co možná nejrůznější vstupní data. Cílem jeho práce je odhalit v programu co nejvíce chyb. Čím dříve se chyba objeví, tím snadnější a levnější je její oprava. Chyby v ostrém provozu u zákazníka jsou ty nejdražší.

Před předáním programu testerovi si základní testování ovšem provádí i programátor. Jako autor programu nejlépe tuší, kde asi mohou být citlivá místa, a tak se je snaží zkontrolovat. **Zvykněte si i vy své programy testovat.** Testujte o to pečlivěji, že vám záda nekryje jiná osoba v podobě testera. Přitom pamatujte, že **cílem testování není najít případ, kdy program funguje, ale především najít případy, kdy program nefunguje!**

SHRNUTÍ KAPITOLY

- Jedním z nejdůležitějších příkazů vůbec je přiřazovací příkaz. V jazyce C# se zapisuje v této podobě:

```
kam = co;
```

  Příkaz funguje tak, že hodnotu uvedenou napravo od rovnítka přiřadí (zkopíruje) do místa zapsaného na straně levé.

- Při práci s vlastnostmi různých objektů se používá bližší určení jménem objektu, které lze schematizovat do sloganu „jméno objektu, tečka, jméno vlastnosti daného objektu“. Například `poleVěta.Text` označuje vlastnost `Text` objektu `poleVěta`.
- Chcete-li v metodě vykonat více akcí, jednoduše zadáte více příkazů. Příkazy se pak vykonávají sekvenčně, jeden za druhým, v pořadí, v jakém jsou ve zdrojovém kódu zapsány.
- Kromě syntaktických chyb, což jsou prohřešky proti pravidlům programovacího jazyka, existují ještě chyby sémantické, jak se označují případy, kdy se program sestaví, ale po spuštění dělá něco jiného, než programátor zamýšlel. Zatímco syntaktické chyby odhalí vývojové prostředí, zůstává odhalení chyb sémantických na člověku. Za tím účelem je žádoucí každý program dostatečně důkladně otestovat.