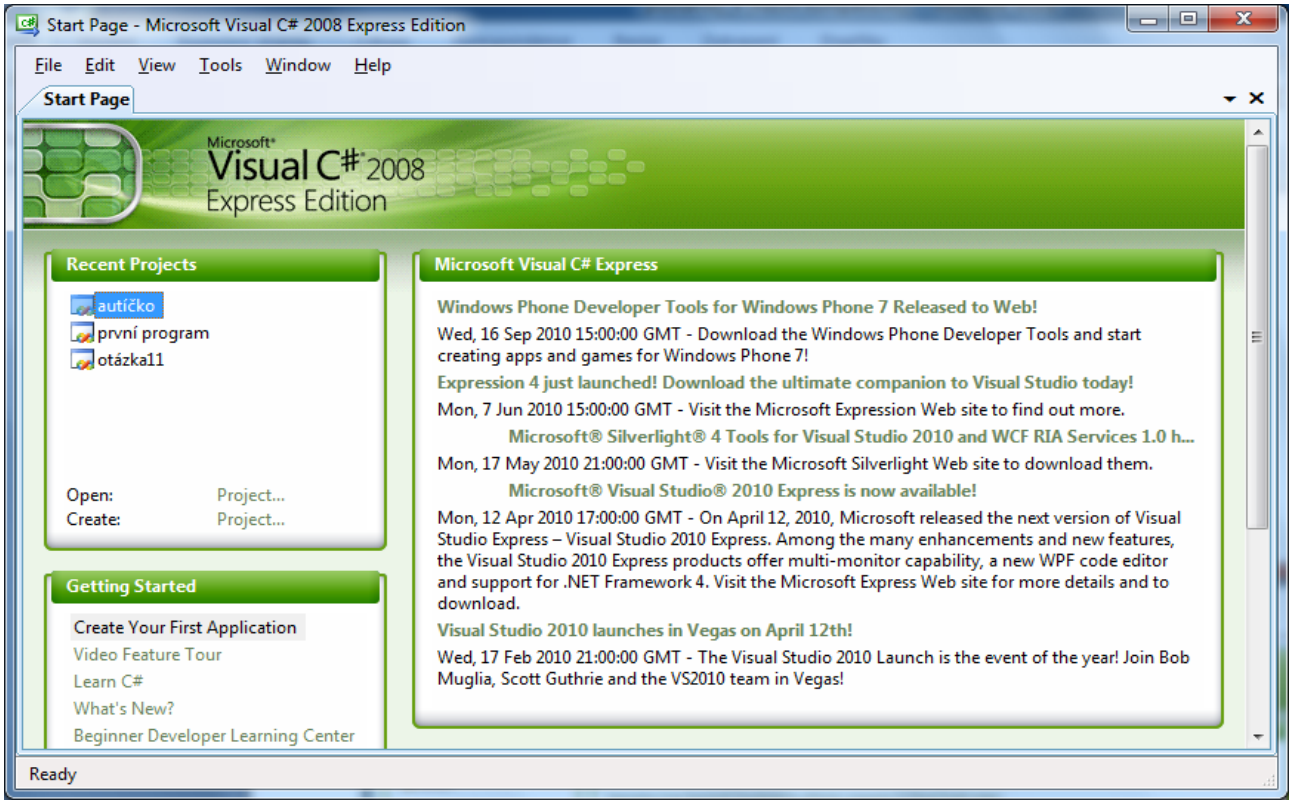
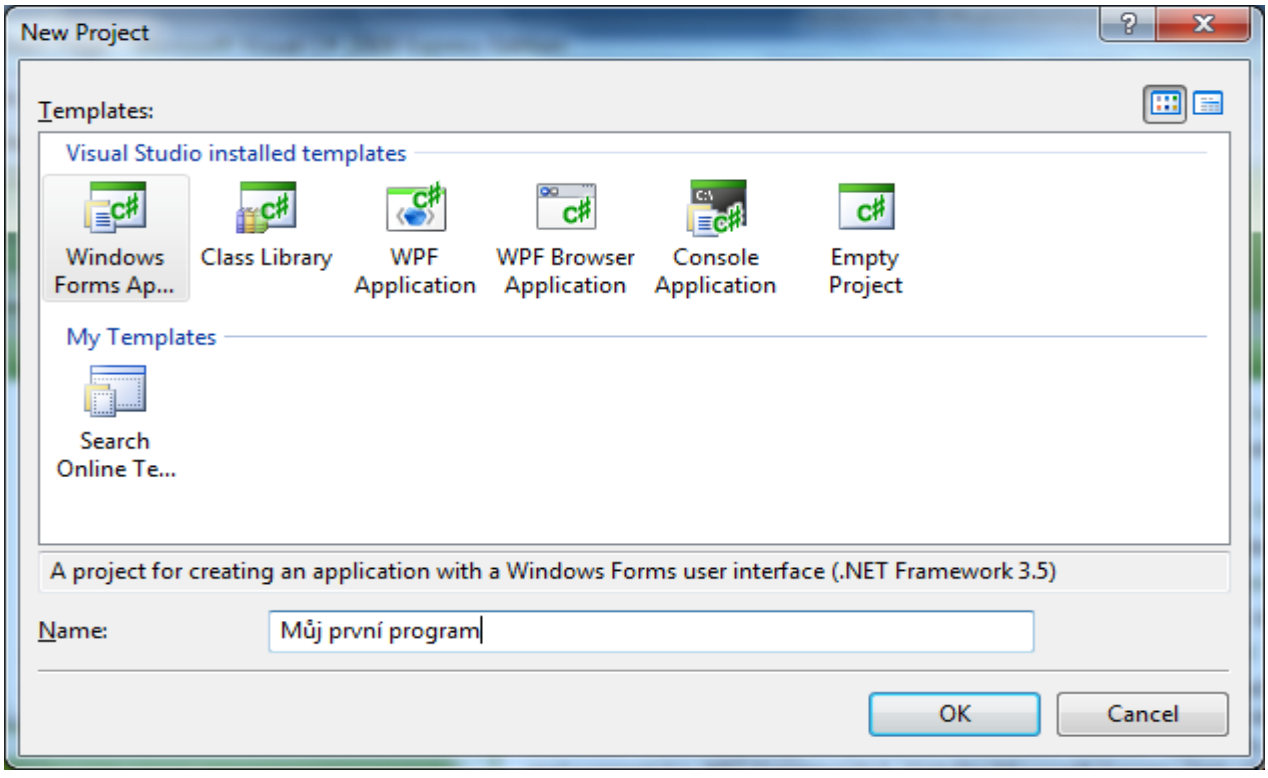
1. Programovací jazyk C# a vývojové prostředí

Po spuštění vývojového prostředí se objeví úvodní obrazovka:



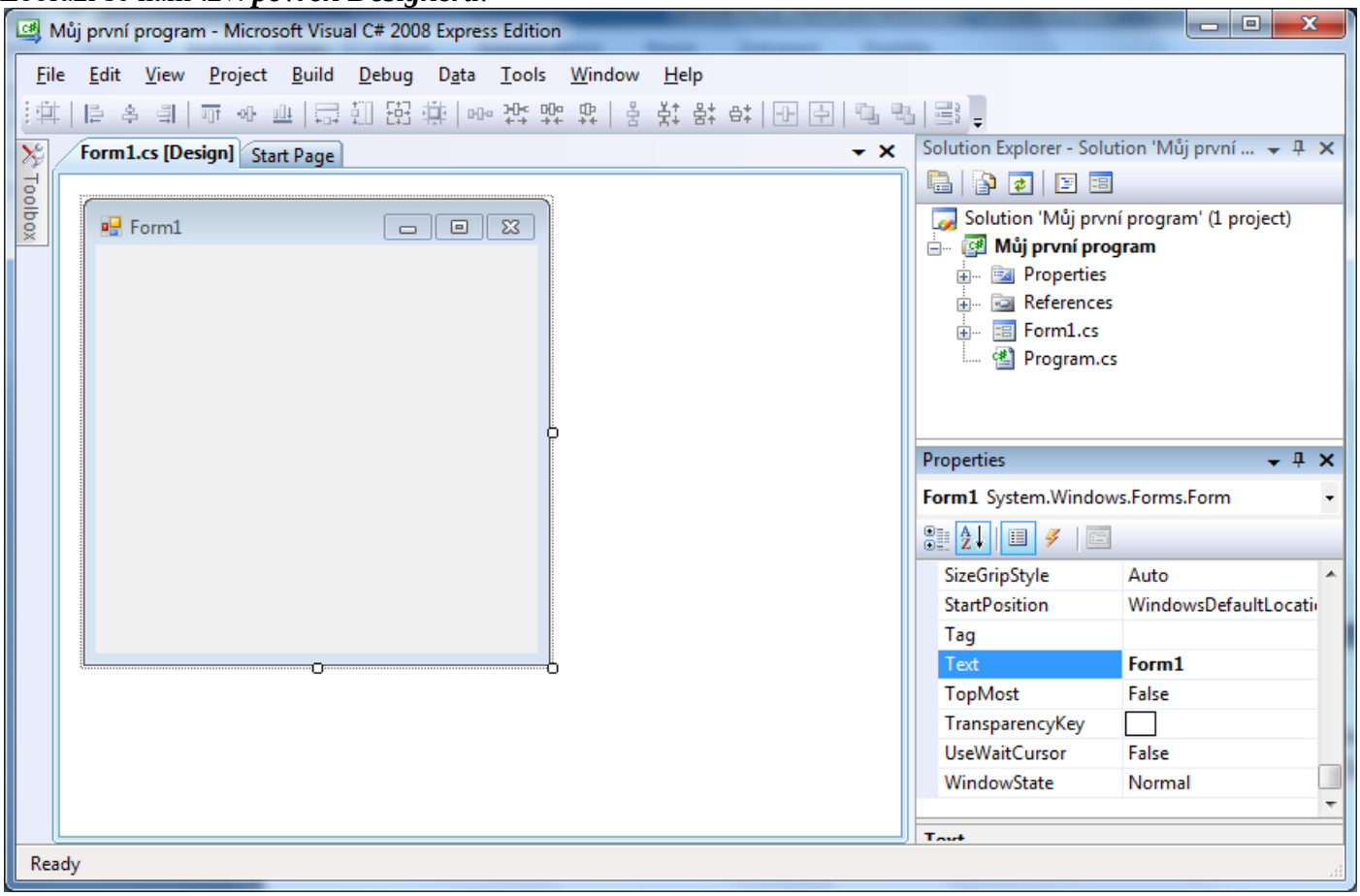
První program

Z nabídky *File* zvolte *New Project*. Otevře se dialogové okno:



Je třeba vybrat vhodnou šablonu podle toho, jaký druh programu budeme vytvářet. My budeme používat aplikace pro Windows s grafickým uživatelským rozhraním, proto vybereme *Windows Forms Application*. V poli *Name:* zadáme jméno nového projektu a potvrdíme tlačítkem *OK*.

Zobrazí se nám tzv. ***povrch Designera***:



Povrch designera ukazuje navrhovaný vzhled programu. Úvodní obrazovka se nám schovala pod záložku *Start Page*.
Vpravo je **průzkumník řešení** (Solution Explorer), v kterém vidíme a můžeme otevírat jednotlivé součásti projektu (designera najdeme pod označením Form1.cs, Program.cs skrývá zdrojový kód programu). Pod ním je **editor vlastností a událostí** (PropertiesWindow), pokud vybereme konkrétní objekt, tak zde uvidíme vlastnosti případně události daného objektu. Vlevo je sada nástrojů (Toolbox), to je panel pro výběr ovládacích prvků. Pokud si některé z těchto oken zavřeme, znovu je otevřeme z nabídky *View*.
Okno, které vidíme v designeru je předlohou okna, v kterém poběží náš projekt. Název okna a jeho další vlastnosti lze měnit v editoru vlastností a událostí. Stačí vybrat okno programu a vpravo uvidíme výpis všech jeho vlastností. Na události přepneme tlačítkem s „bleskem“, ale to až později.
Vlastností je velké množství, stačí znát jen ty nejpoužívanější. U okna programu budeme vždy měnit název a text v titulní liště.
Název je vlastnost `(Name)` – automaticky tam bude Form1, my přepíšeme např. na OknoProgramu.
U názvu objektu musí být jedno slovo bez mezer. Pro oddělení slov použijeme velká písmena. Češtinu (háčky, čárky) používat můžeme.
Text v titulní liště je vlastnost `Text`. Tam napíšeme přesně to, co bude na titulní liště okna programu, samozřejmě česky a s mezerami.
Další často používané vlastnosti jsou:

- `BackColor` – barva pozadí okna
- `FormBorderStyle` – styl rámečku (např. jestli lze měnit velikost okna apod.)
- `MinimeBox, MaximeBox`– jestli má ikonku pro minimalizaci nebo maximalizaci
- `Size` – výchozí velikost okna

I když náš program ještě „nic nedělá“, můžeme ho spustit

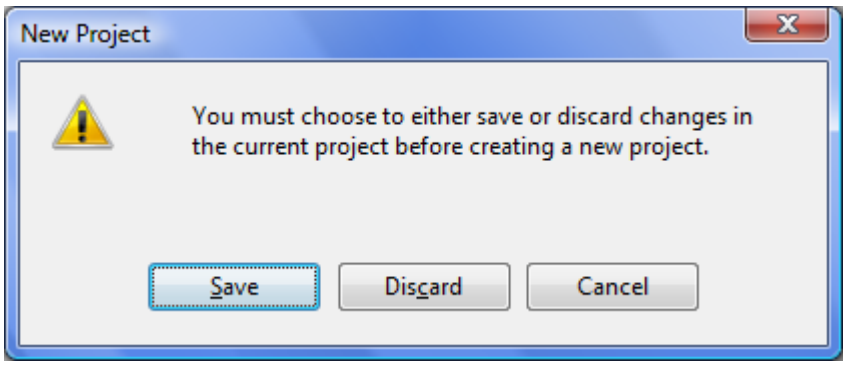
Spuštění programu

Program spustíme z nabídky *Debut - Start Debugging* nebo klávesou F5.

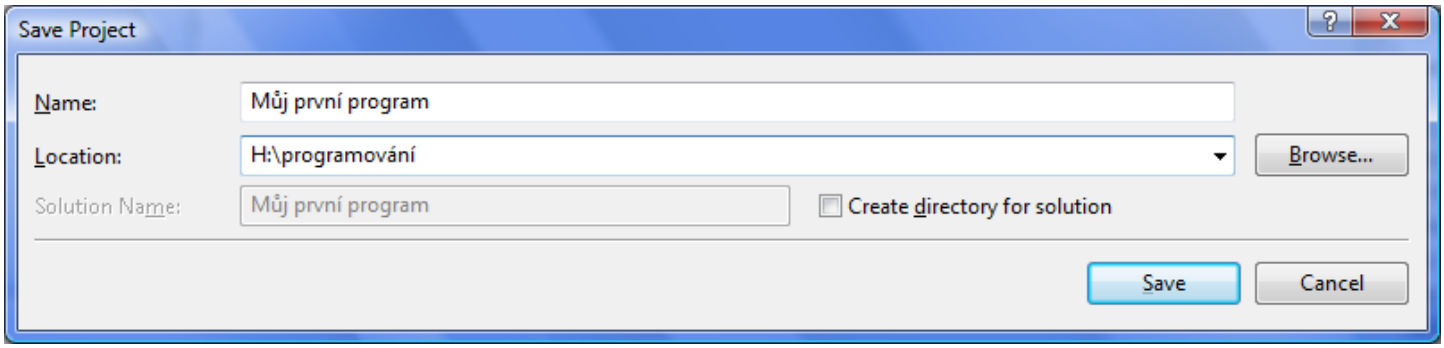
Po spuštění se objeví stejné okno, jako jsme viděli v designeru. Rozdíl mezi nimi je ten, že okno v designeru je „neživá předloha“, jak bude program vypadat. Spuštěný program je skutečný, „živý“. Je to samostatně existující okno, které lze zvětšovat, přemisťovat, atd. Ovládací prvky umístěné v tomto okně (až tam nějaké budou) fungují – tlačítka reagují na stisk, do textových polí lze psát, atd.
Než budeme pokračovat v sestavování programu v designeru, **musíme spuštěný program uzavřít křížkem.**

Uložení programu

Každý projekt bychom měli uložit. Buď vyčkáme, až budeme dotázáni, zda chceme projekt uložit – při zavírání vývojového prostředí nebo před spuštěním nového projektu, na dotaz odpovíme *Save* nebo uložíme sami z nabídky *File – SaveAll*. Při prvním ukládání se program ptá na umístění, pak už ukládá bez ptaní do stejného místa. Program je lepší ukládat průběžně. **K ukládání projektu nepoužívejte *Save As***, uložíte pouze jeden soubor, ale ne celý projekt!



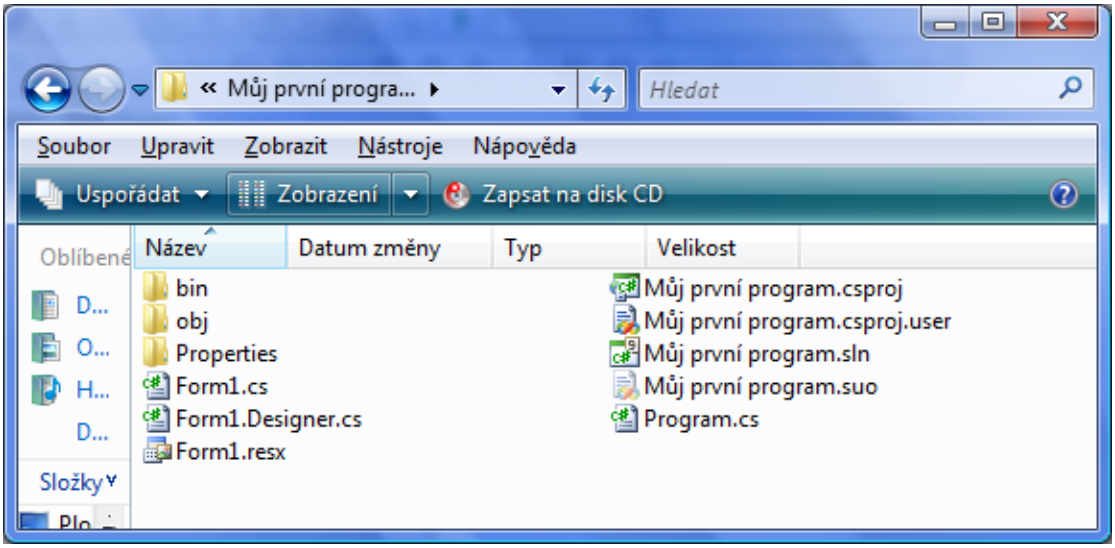
Dotaz na uložení změn



Dotaz na místo uložení

Při zadávání místa uložení zrušíme zaškrtnutí políčka *Create direktory forsolution.*

Soubory projektu



Při ukládání projektu vznikne složka s názvem z řádku *Solution Name* a do ní se uloží všechny soubory projektu. Soubory s příponou `.cs.` jsou zdrojové texty programu, obsahují vlastní zápis programu v jazyce C#. Jsou ve formátu prostého textu a lze je editovat např. v poznámkovém bloku.
Dále se ve složce nachází soubor s příponou `.sln` a `.csproj.` Pro začátečníka v programování není nutné vědět, k čemu slouží, stačí si pamatovat, že do nich nebudeme zasahovat.

Při kopírování projektu přeneseme vždy celou složku. Pro spuštění projektu poklepeme buď na soubor `.sln` nebo `.csproj` a projekt se otevře ve vývojovém prostředí.
Ve složce projektu bude podsložka *bin* a v ní podsložka *debug*. Tam najdeme spustitelnou verzi projektu (aplikaci), která bude fungovat bez vývojového prostředí. Vznikne samozřejmě až po té, co projekt úspěšné spustíme. Obsahuje naposledy spuštěnou verzi.

2. Ovládací prvky

Ovládací prvky budeme do okna programu přetahovat z toolboxu (vlevo od okna programu). Prvky jsou uspořádány do skupin. Zpočátku budeme používat hlavně skupinu *Common Controls*.
Daný prvek zvolíme v toolboxu a do okna programu vložíme tažením levým tlačítkem myši. Stejný prvek můžeme dále vkládat kopírováním (Ctrl+c, Ctrl+v).
U všech prvků zadáme jejich název (Name). Pravidla pro zadávání názvů již známe od okna programu. Je vhodné prvky přejmenovávat a volit pojmenování, která napovídají, k čemu daný prvek slouží.
Při změně vlastností je třeba daný objekt vybrat myší, ale pozor, **vždy jen jedním kliknutím**. Pokud kliknete dvakrát, vyvoláte pro daný objekt obslužnou metodu hlavní události a otevře se vám okno, do kterého budete tuto metodu zapisovat. Toto okno je třeba zavřít a v editoru vlastností přepnout na události (tlačítko s bleskem). Danou událost najdeme, stiskneme na ní pravé tlačítko a vybereme *reset*. Pak je naše chyba napravená. **Nikdy sami nemažeme vložený kód prázdné metody**.
Následující ovládací prvky si vyzkoušíme na programu *Osobní údaje*

Textové pole - TextBox

Textové pole využíváme jak pro zadávání vstupních údajů, tak pro výpis výstupních údajů.
Vlastnosti textového pole:

`Text`	text v textovém poli
`BackColor`	barva pozadí textového pole
`ForeColor`	barva textu v textovém poli
`Font`	druh a velikost použitého písma
`MaxLength`	maximální počet znaků, které může uživatel do pole zadat
`Multiline`	při nastavení `True` umožňuje zadání víceřádkového textu
`ReadOnly`	při nastavení `True`není možné do pole zapisovat, ale pole je aktivní
`Enabled`	při nastavení `False` je pole neaktivní
`ScrollBars`	umožňuje okolo pole zobrazit posuvníky
`TextAlign`	zarovnání textu v poli
`UseSystemPasswordChar`	při nastavení `True` místo znaků zobrazuje puntíky
`WordWrap`	při nastavení `True` automaticky zalamuje text (musí být `Multiline`)

Popisek - Label

Popisek je nápis, který se nejčastěji umísťuje v blízkosti jiných ovládacích prvků a tím naznačuje jejich význam.
Vlastnosti popisku: `Text, backColor, ForeColor, Font`

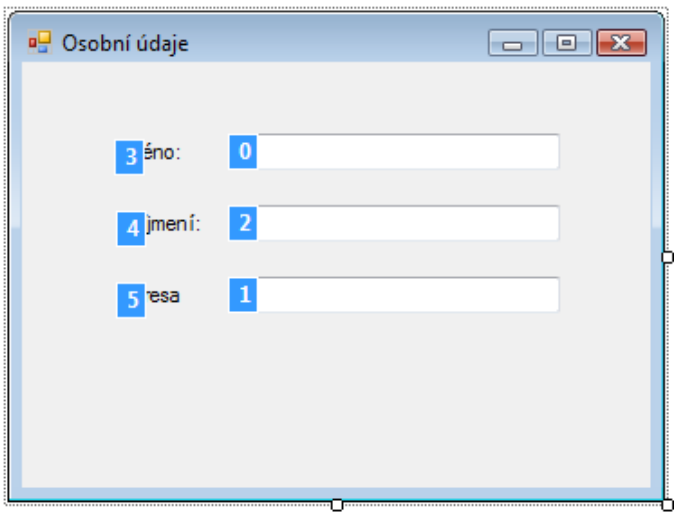
Zarovnání ovládacích prvků

Pro zarovnání ve vodorovném směru využijeme modré vodící čáry, kterénám vývojové prostředí samo nabízí. Pro vytvoření stejných mezer mezi prvky ve svislém směru musíme tyto prvky označit (pomocí klávesy Ctrl) a poté použít nabídku *Format – VerticalSpacing – Make Equal*
Pro vycentrování prvku na stránce použijeme nabídku *Format – Center in Form*.

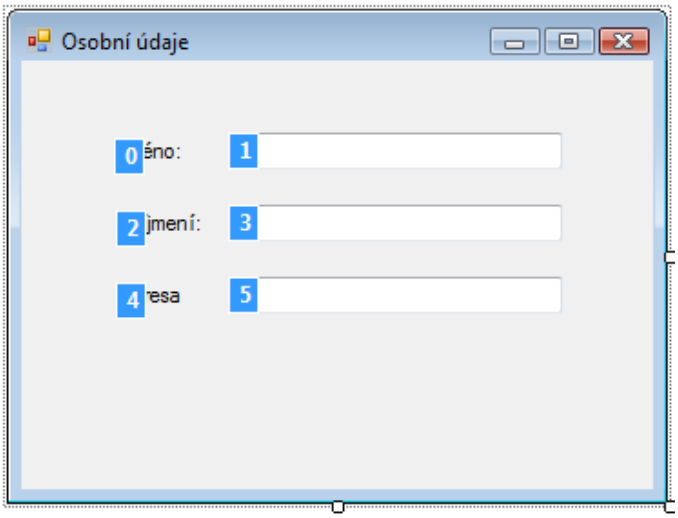
Klávesové zkratky a pořadí tabulátoru

Popisku můžeme přiřadit klávesovou zkratku *Alt+písmeno*. Stačí do vlastnosti `text` napsat před dané písmeno znak &. Daný znak se v popisku podtrhne (někdy se podtržení objeví až po stisku klávesy *Alt*, záleží na nastavení Windows).

např. Objekt má jméno PopisekJméno a text &Jméno. Po spuštění programu pak stačí stisknout *alt+j* a kurzor skočí do textového pole za popiskem.
Aby toto fungovalo, musíme mít v pořádku pořadí tabulátoru, tj. pořadí v jakém se mi budou postupně aktivovat jednotlivé prvky při mačkání klávesy *Tab*. U každého prvku je vlastnost `TabIndex`, která určuje toto pořadí. Tento index se přiděluje podle pořadí, v jakém prvky do okna umísťujeme. Indexuje se od nuly. Abychom nemuseli od počátku myslet na toto pořadí nebo ho později nepřepisovali každému prvku zvlášť, můžeme využít nabídku *View – TabOrder* a zobrazit si tak pořadí tabulátoru. Klikáním na jednotlivá čísla hodnoty správně přečíslujeme. Čísla skryjeme tím, že znovu zvolíme *View – TabOrder*.



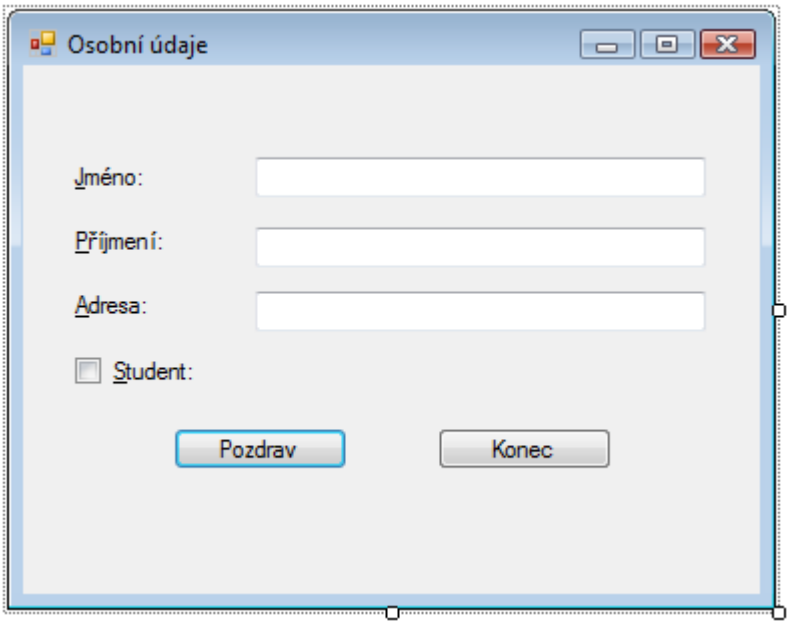
Původní pořadí tabulátoru



Správné pořadí tabulátoru

Zaškrtávací políčko - CheckBox

Zaškrtávací políčko je ovládací prvek pro dvoustavovou informaci ano/ne. Vlastnost `Text` určuje text popisku vedle zaškrtávacího políčka. Vlastnost `Checked` určuje zda je políčko zaškrtnuté (hodnota `True`) nebo není zaškrtnuté (hodnota `False`).



Tlačítko – Button

Tlačítko je zaoblený obdélník s textem, který po stisknutí spustí nějakou akci.
Vlastnost `Text` udává text napsaný na tlačítku.
Náš program *Osobní údaje*vypadá jako na následujícím obrázku.
Pořadí tabulátoru je v pořádku, textová pole jsou aktivní, klávesové zkratky fungují, pouze tlačítko zatím po stisku nic nevyvolává. Abychom ho „oživili, musíme umět napsat obslužný program pro událost *Click*.“

3. Události

Běh programu je řízen událostmi. Událostí může být klepnutí myší, přesunutí myši, stisk nebo uvolnění klávesy, načtení okna, impuls časovače, atd.
Budeme pokračovat v práci na našem programu *Osobní údaje* tím, že oživíme jeho tlačítka.

Událost Click

Nejdůležitější událostí tlačítka je událost `Click`, která vznikne při jeho stisknutí. Aby na ní program reagoval, musíme k ní připojit obslužný podprogram(metodu):

- vybereme příslušné tlačítko
- v editoru vlastností a událostí přepneme na události – blesk

- poklepeme na příslušnou událost, tj. `Click`
- otevře se zdrojový kód `Form1.cs` a vloží se do něj prázdnou metodu
- v místě, kde bliká kurzor, mezi složenými závorkami, píšeme

Pro tlačítko *Konec* napíšeme metodu, která zařídí, že po stisku tlačítka se ukončí program. Provedeme celý výše uvedený postup. Vývojové prostředí za nás připraví zápis metody včetně hlavičky –
`privatevoid TlačítkoKonec_Click(object sender, EventArgs e)`
Za hlavičkou následuje tělo (mezi složenými závorkami). Do těla zapíšeme příkaz `Close();`
Použili jsme podprogram `Close` pro ukončení programu. Za názvem následují vždy závorky, v kterých jsou někdy uvedené parametry pro volání podprogramu. Tento příkaz žádné parametry nemá, proto jsou závorky prázdné.

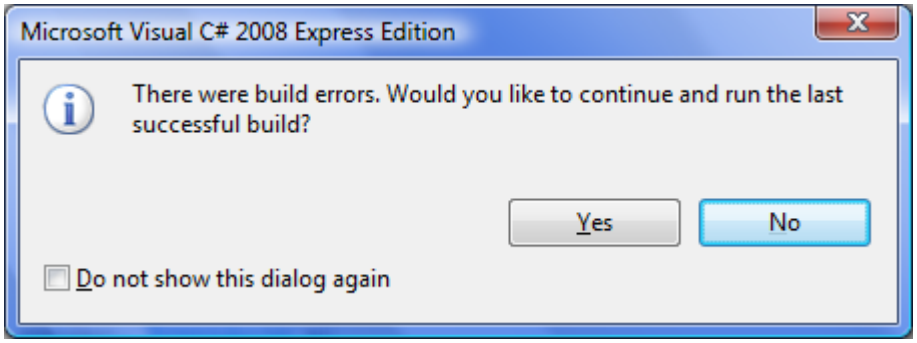
Podprogram `Close` už před námi někdo naprogramoval a mi jej můžeme používat díky tomu, že máme připojenu tu správnou knihovnu podprogramů. Některé základní knihovny nám připojí vývojové prostředí samo. Pokud bychom chtěli využívat i jiných knihoven, museli bychom si je připsat. Výpis připojených knihoven najdete na začátku v okně `Form1.cs`, to co je uvedeno slovem `using`. Naše `Close` je z knihovny `System.Windows.Forms`.

Pro tlačítko *Pozdrav* zopakujeme celý postup, ale místo `Close();` napíšeme
`MessageBox.Show("Dobrý den!");`
Po stisku tlačítka se objeví okno s textem „Dobrý den! “ a potvrzovacím tlačítkem, kterým toto okno zavřeme.
Text v závorce je v uvozovkách, tím jsme dali najevo, že jde o textový řetězec.
Při zápisu podprogramu využíváme toho, že nám vývojové prostředí našeptává co máme psát. Díky tomu nemusíme znát přesný zápis názvů používaných podprogramů ani přesné názvy naších vlastních objektů.
Každý příkaz musí být ukončen středníkem.

Výpis `Form1.cs`:

```
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;

namespace Osobní_údaje
{
    public partial class OknoProgramu : Form
    {
        public OknoProgramu()
        {
            InitializeComponent();
        }

        private void TlačítkoKonec_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void TlačítkoPozdrav_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Dobrý den!");
        }
    }
}
```

Chyba v programu



Pokud je v programu nějaká chyba, program se nespustí a objeví se následující okno

Na tuto hlášku vždy odpovězte „No“. Není nutné pouštět jiný program.
Po uzavření hlášky se ve spodní části okna objeví „Error list“ – výpis chyb. Nejdůležitější chyba je vždy ta první.

Odstraňujeme vždy nejdříve první chybu.

Další události

Objekt může mít napsané metody i pro více událostí najednou.
Tlačítku *Konec* přiřadíme událost `MouseHover`. Tato událost nastane, když uživatel najede nad tlačítko myší a tam zastaví. Jako reakci na tuto událost vyvoláme okno se zprávou „Myš se zastavila nad tlačítkem“.
Do těla události `TlačítkoKonec_MouseHover` napíšeme :

```
MessageBox.Show("Myš se zastavila nad tlačítkem");
```

Společná obsluha více událostí, přejmenování

Pokud stejnou metodu budeme chtít použít i pro druhé tlačítko, je zbytečné i nepraktické psát ji znovu. Stačí v událostech vybrat stejnou událost (MouseHover) a jedním kliknutím zobrazit vpravo šipku. Šipku rozbalíme a vybereme si z již nadefinovaných metod.
Poté co metoda `TlačítkoKonec_MouseHover` obsluhuje obě tlačítka, je vhodné ji přejmenovat, např. `Tlačítka_MouseHover`. Přejmenování provedeme pod dohledem vývojového prostředí. V `Form1.cs` najdeme název metody, klikneme na něj pravým tlačítkem a zvolíme *Refactor – Rename*. Do pole *New name* zadáme nové jméno, potvrdíme *OK* a v dalším okně potvrdíme *Apply*.

Hlavní událost objektu

Každý objekt má svou hlavní událost. Obslužnou metodu této události můžeme vyvolat i tím, že na objekt dvakrát klikneme. Co dělat, když se nám to povede nechtěně, jsme již řešili dříve.

Objekt	*Hlavní událost*	*Kdy událost vzniká*
Okno programu	`Load`	Těsně před zobrazením okna programu
Textové pole	`TextChanged`	Když se změní text v poli
Popisek	`Click`	Když uživatel klikne na popisek
Tlačítko	`Click`	Když uživatel stiskne tlačítko
Zaškrtávací políčko	`CheckedChanged`	Když se změní stav zaškrtnutí

Obsluha tlačítek z klávesnice

Naše dvě tlačítka ještě vylepšíme tak, že jedno bude reagovat na stisk klávesy *Enter* a druhé na *Esc*. Vybereme okno programu a v jeho vlastnostech nastavíme pro vlastost *AcceptButton* TlačítkoPozdrav a pro vlastnost *CancelButton* TlačítkoKonec. Metoda pro obsluhu tlačítka Konec se pak vyvolá i klávesou *Esc* a pro obsluhu Pozdrav se vyvolá klávesou *Enter*.

4. Další ovládací prvky

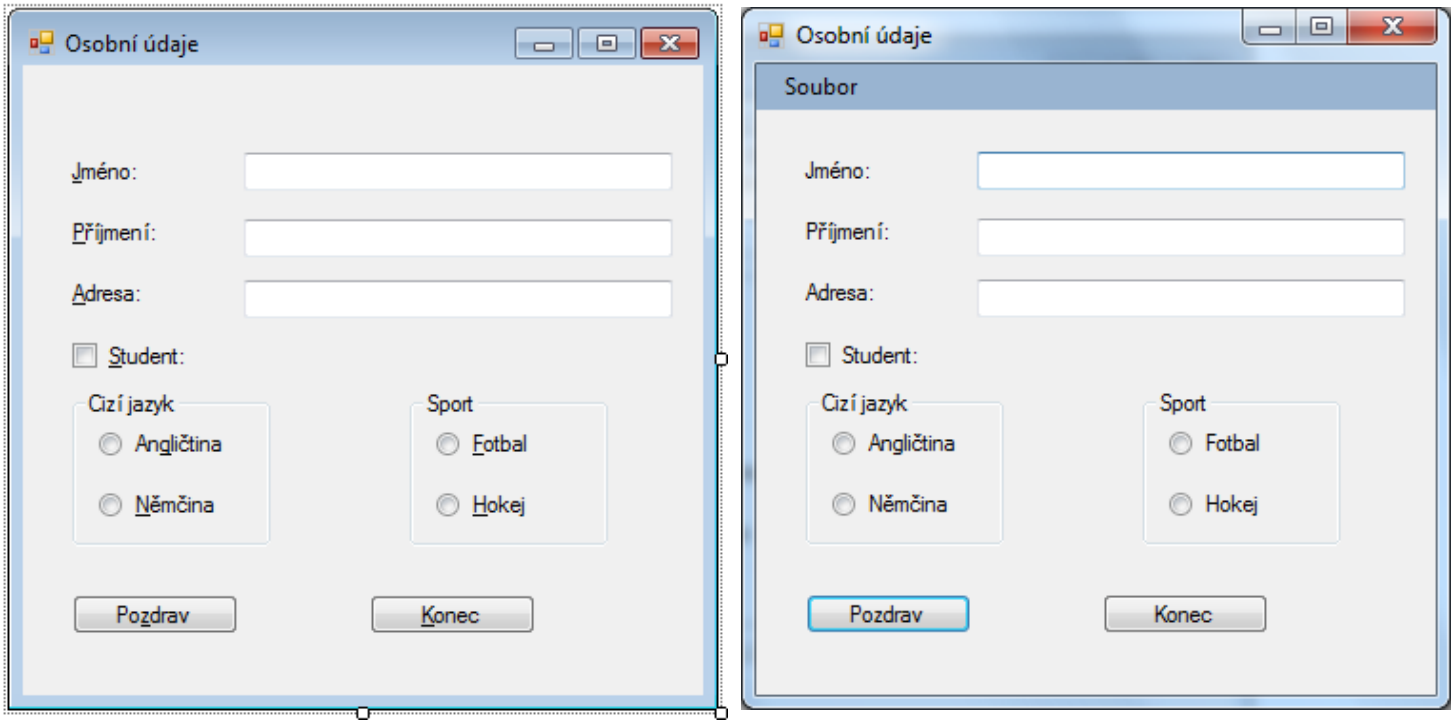
Přepínač – RadioButton, skupina - GroupBox

Přepínač má podobnou funkci jako zaškrtávací políčko, slouží pro vkládání dvoustavové informace ano/ne. Rozdíl je v tom, že přepínače se sdružují do skupin. Ve skupině může být zaškrtnuta jen jedna možnost. Skupinu, která sdružuje dané přepínače najdeme v toolboxu v kategorii *Cointainers* pod názvem *GroupBox*.

Jednotlivé přepínače pak umístíme na tento kontejner a najdeme je v kategorii *Common Controls* pod názvem RadioButton.
Náš program *Osobní údaje* si rozšíříme o dvě skupiny přepínačů:

Nabídka – MenuStrip

Nabídku najdeme v toolboxu v kategorii Menus&Toolbars pod názvem MenuStrip.
Po přetažení do designera se objeví nápis Type here, který znamená, že můžeme zadávat položku nabídky.
Oddělovač položek získáme zadáním pomlčky místo textu.
Hlavní událostí je Click. Pokud chceme z nabídky ovládat stejnou funkci jako třeba z již nadefinovaného tlačítka, stačí vybrat správnou metodu a nemusíme ji znovu psát.
Podtržené znaky se zadají znakem &. Přidání další klávesové zkratky lze provést zadáním vlastnosti ShortCutKeys.
Náš program Osobní údaje rozšíříme o nabídku.

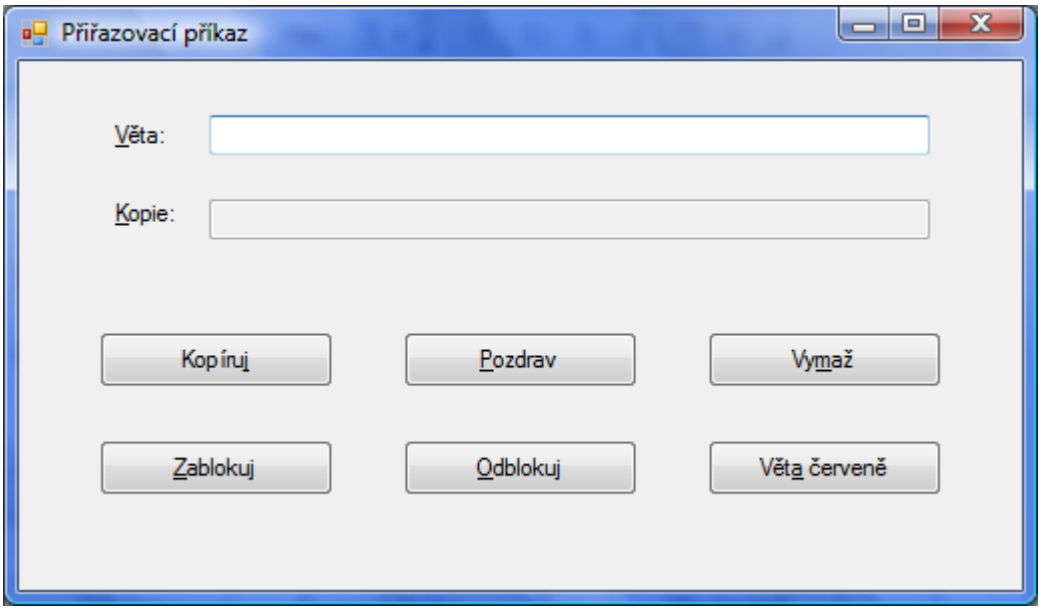


5. Přiřazovací příkaz

Připravíme uživatelské rozhraní pro další program podle obrázku:

Nezapomeneme na klávesové zkratky Alt+písmeno a na pořadí tabulátoru.

Pro obsluhu události `click` použijeme u všech tlačítek ***přiřazovací příkaz***, který **hodnotu na pravé straně přiřadí na místo dané na levé straně.**



```
privatevoid TlačítkoKopíruj_Click(object sender, EventArgs e)
        {
            PoleKopie.Text = PoleVěta.Text;
        }

privatevoid TlačítkoPozdrav_Click(object sender, EventArgs e)
        {
            PoleVěta.Text = "Ahoj lidi!";
        }

privatevoid TlačítkoVymaž_Click(object sender, EventArgs e)
        {
            PoleVěta.Text = "";
        }

privatevoid TlačítkoČervená_Click(object sender, EventArgs e)
        {
            PoleVěta.ForeColor = Color.Red;
        }

privatevoid TlačítkoZablokuj_Click(object sender, EventArgs e)
        {
            PoleVěta.Enabled = false;
        }

privatevoid TlačítkoOdblokuj_Click(object sender, EventArgs e)
        {
            PoleVěta.Enabled = true;
        }
```

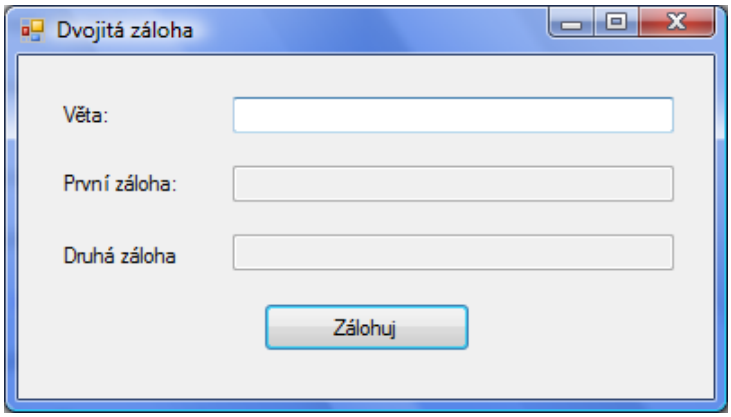
Více příkazů v metodě

Dosud jsme měli v každé metodě pouze jeden příkaz. Nemusí to tak být vždy.
Vylepšíme si tlačítka *Zablokuj* a *Odblokuj*. Tlačítko, které v danou chvíli nemá smysl, bude zablokované.

```
privatevoid TlačítkoZablokuj_Click(object sender, EventArgs e)
        {
            PoleVěta.Enabled = false;
            TlačítkoZablokuj.Enabled = false;
            TlačítkoOdblokuj.Enabled = true;
        }

privatevoid TlačítkoOdblokuj_Click(object sender, EventArgs e)
        {
            PoleVěta.Enabled = true;
            TlačítkoZablokuj.Enabled = true;
            TlačítkoOdblokuj.Enabled = false;
    }
```

Sekvenční zpracování

Připravíme uživatelské rozhraní pro další program podle obrázku:



Pokud napíšeme obslužnou metodu události `click` tlačítka *Kopíruj* následujícím způsobem, zjistíme, že nejde o dvojitou zálohu, ale do obou polí se kopíruje totéž.

```
privatevoid TlačítkoZálohuj_Click(object sender, EventArgs e)
        {
            PolePrvníZáloha.Text = PoleVěta.Text;
            PoleDruháZáloha.Text = PolePrvníZáloha.Text;
        }
```

Aby se do druhé zálohy kopíroval obsah první, musíme příkazy přehodit.

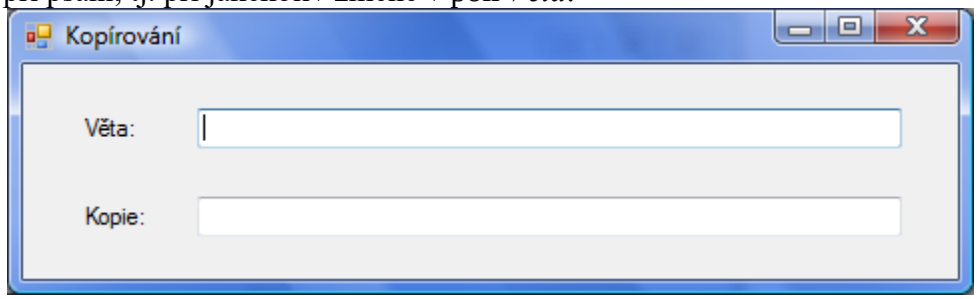
```
privatevoid TlačítkoZálohuj_Click(object sender, EventArgs e)
        {
            PoleDruháZáloha.Text = PolePrvníZáloha.Text;
            PolePrvníZáloha.Text = PoleVěta.Text;
        }
```

Příkazy v metodě se provádí sekvenčně, tj. jeden za druhým v pořadí, jak jsou napsány.

Příklady na procvičení:

Příklad 1: Do programu *přiřazovací příkaz* přidejte tlačítko *Obnov původní stav*, které nastaví zpět výchozí barvu textu, vymaže obě textová pole, zablokuje tlačítko *Odblokuj* a odblokuje tlačítko *Zablokuj*.

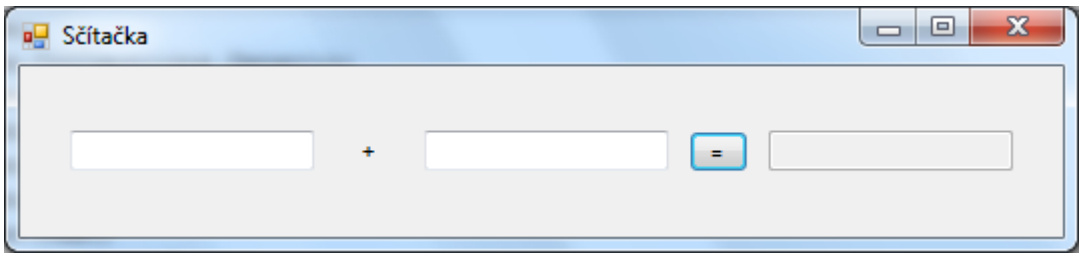
Příklad 2: Napište program *Okamžitá kopie* podle následujícího obrázku. Kopie věty se bude vytvářet hned při psaní, tj. při jakékoliv změně v poli *Věta*.



Příklad 3: Napište program *Schovávaná* s jedním tlačítkem *Schovej se* , které schová celý program na dvě vteřiny. Použijte příkazy `Hide();` a `Show();` a na „uspání“ programu na dvě vteřiny použijte `System.Threading.Thread.Sleep(2000);`
Program dále vylepšete tak, aby tlačítko bylo žluté a zčervenalo, když nad ním bude myš. Použijte události *MouseEnter* a *MouseLeave*
Dále přidejte tlačítko *Pípej*, které vyvolá pípnutí, viz příkaz `System.Media.SystemSounds.Beep.Play();`

6. Typy a proměnné

Připravíme si uživatelské rozhraní podle následujícího obrázku. Textová pole nazveme PoleČíslo1, PoleČíslo2, PoleVýsledek. Tlačítko bude TlačítkoRovnáSe.



Pro tlačítko napíšeme obslužnou metodu události `Click`.

```
privatevoid TlačítkoRovnáSe_Click(object sender, EventArgs e)
        {
            PoleVýsledek.Text = PoleČíslo1.Text + PoleČíslo2.Text;
        }
```

Po spuštění programu zjistíme, že sčítačka nesčítá, ale spojuje, a to jak čísla, tak řetězce znaků.

Datové typy

Při zadávání hodnot vlastností jednotlivých objektů, jsme používali různé druhy hodnot (např. čísla textové řetězce, logické hodnoty – true/false).
Takto chápanému druhu hodnot říkáme **datové typy**. Datový typ udává, kde můžeme danou hodnotu použít a jaké operace s ní můžeme provádět.

Dosud jsme používali následující typy:

Druh hodnoty	*Název typu*
Textový řetězec	string (String)
Barva	Color
Logická hodnota	bool (Boolean)
Celé číslo	int (int32)

V závorkách jsou uvedené alternativní názvy, které nemusíme používat, ale je dobré o nich vědět.

Pro správné fungování naší sčítačky potřebujeme převést číslo na řetězec a obráceně. K tomu potřebujeme typovou konverzi.
Z řetězce na číslo: ConvertToInt32(*převáděný řetězec*);
Z čísla na řetězec: ConvertToString(*převáděné číslo*);

Proměnné

Ze zadaného textu v PoleČíslo1 a PoleČíslo2, už umíme datovou konverzí vyrobit číslo. Abychom s ním mohli dále pracovat, musíme si ho někam uložit, nejlépe do operační paměti. K tomuto účelu potřebujeme proměnnou. Před použitím, musíme proměnnou deklarovat. Deklarací proměnné si program vyhradí v paměti místo, potřebné k uložení hodnoty této proměnné. Deklarace se provede uvedením typu a poté jména proměnné. Deklarace se ukončuje středníkem.

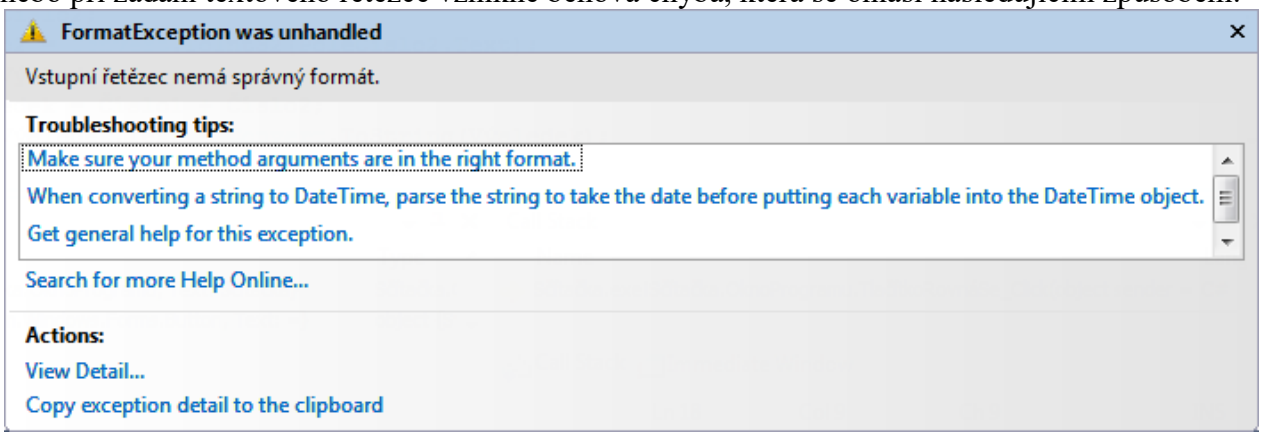
typ_proměnnéjméno_proměnné ;

Nový zápis obslužné metody pro tlačítko = je s použitím proměnných *Číslo1, Číslo2* a *Výsledek*.

```
privatevoid TlačítkoRovnáSe_Click(object sender, EventArgs e)
{
int Číslo1;
   Číslo1 = Convert.ToInt32(PoleČíslo1.Text);
int Číslo2;
   Číslo2 = Convert.ToInt32(PoleČíslo2.Text);
int Výsledek;
   Výsledek = Číslo1 + Číslo2;
   PoleVýsledek.Text = Convert.ToString(Výsledek);
}
```

Běhové chyby

Nyní sčítačka opravdu sčítá, ale pouze pokud do obou polí zadáme celé číslo. Při zadání čísla desetinného nebo pří zadání textového řetězce vznikne běhová chyba, která se ohlásí následujícím způsobem:



Ve zdrojovém kódu programu se žlutě označí místo chyby a k němu výše uvedené okno.
Program musíme ukončit.

Reakcí na běhovou chybu může být zobrazení vhodné hlášky a pokračování v běhu programu, tímto způsobem lze reagovat i na jiné běhové chyby. Tohoto řešení dosáhneme použitím konstrukce `try-catch`.

```
privatevoid TlačítkoRovnáSe_Click(object sender, EventArgs e)
{
      try
      {
            int Číslo1;
            Číslo1 = Convert.ToInt32(PoleČíslo1.Text);
            int Číslo2;
            Číslo2 = Convert.ToInt32(PoleČíslo2.Text);
            int Výsledek;
            Výsledek = Číslo1 + Číslo2;
            PoleVýsledek.Text = Convert.ToString(Výsledek);
      }
catch
      {
      MessageBox.Show("Zadání chybného údaje");
      }
}
```

Příkazy uvedené ve složených závorkách za slovem `try` se provedou pouze, je-li vše „v pořádku“. Pokud v této části dojde k běhové chybě, provede se to, co je uvedeno vesložených závorkách za slovem `catch`.

Deklarace s inicializací

Pokud do proměnné zadáváme počáteční hodnotu hned po deklaraci, lze použít zkrácený zápis:

```
int Číslo1 = Convert.ToInt32(PoleČíslo1.Text);
```

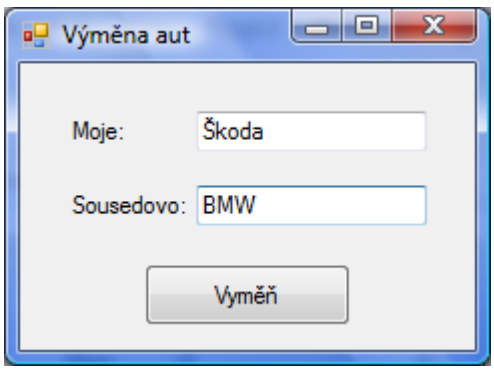
Finální podoba metody `TlačítkoRovnáSe_Click`:

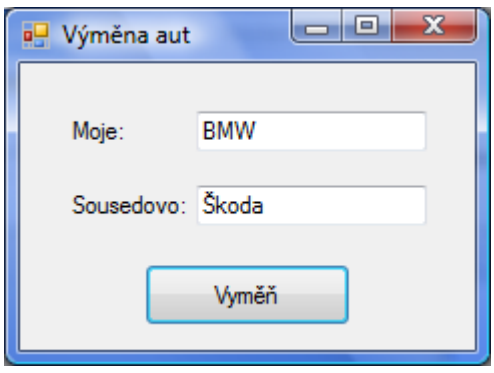
```
privatevoid TlačítkoRovnáSe_Click(object sender, EventArgs e)
{
      try
       {
            int Číslo1 = Convert.ToInt32(PoleČíslo1.Text);
            int Číslo2 = Convert.ToInt32(PoleČíslo2.Text);
            int Výsledek = Číslo1 + Číslo2;
            PoleVýsledek.Text = Convert.ToString(Výsledek);
      }
catch
      {
            MessageBox.Show("Zadání chybného údaje");
      }
}
```

Příklady na procvičení:

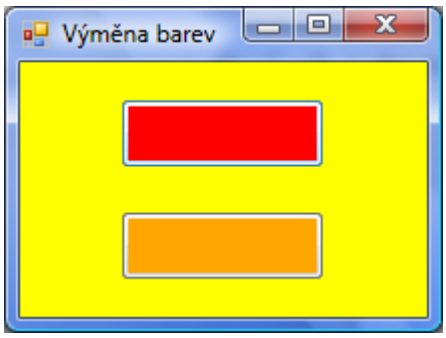
Příklad 1: Napište program na výměnu dvou hodnot mezi sebou.

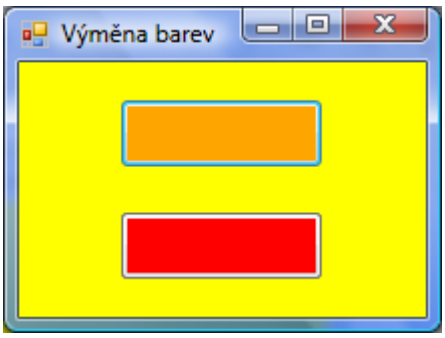
a) Program na výměnu dvou textových řetězců - *Výměna aut*



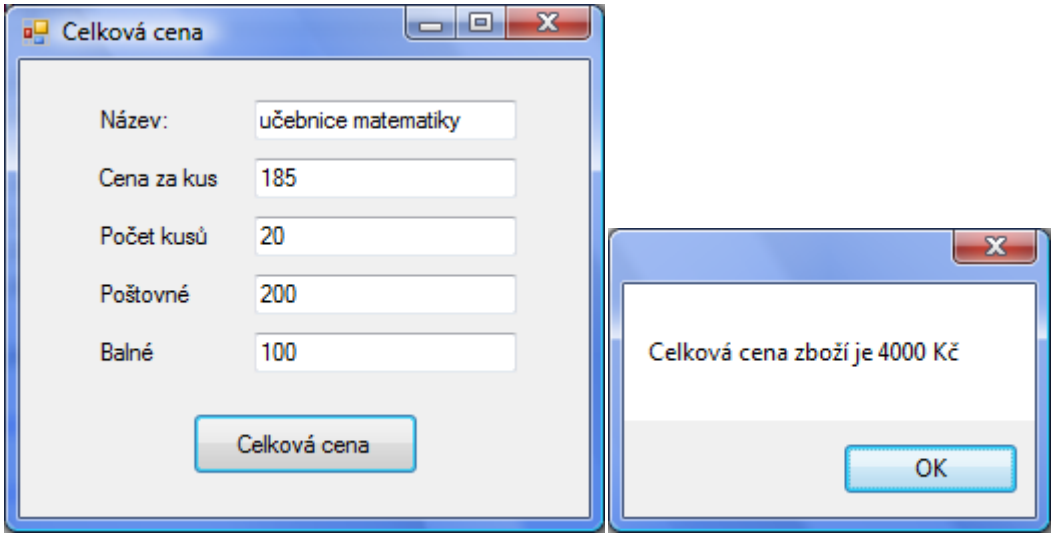


b) Program na výměnu barvy dvou tlačítek – *Výměna barev*

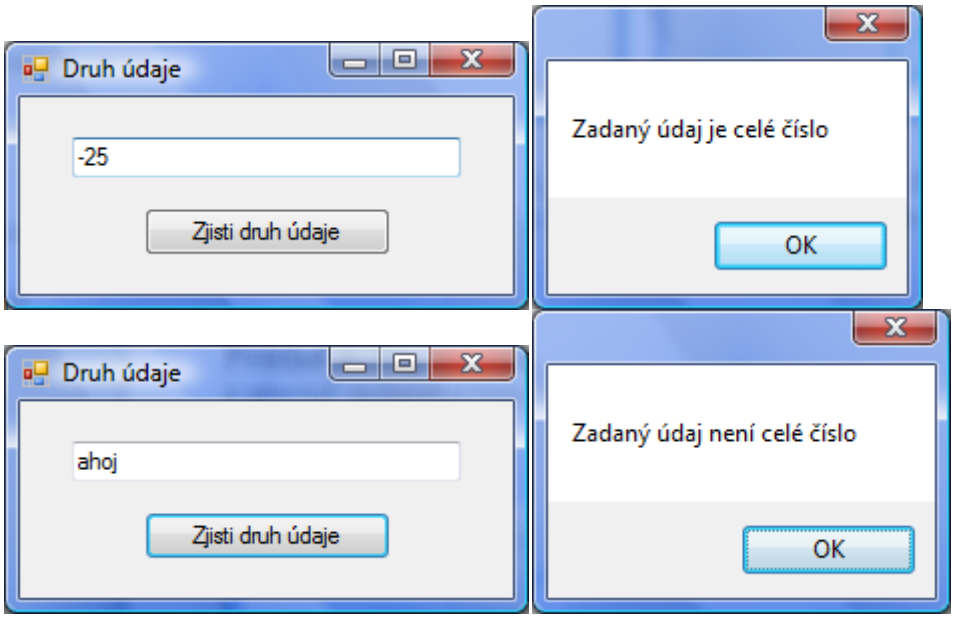




Příklad 2: Napište program, který vypočete celkovou cenu zboží a výsledek zobrazí v okně se zprávou. Pro text výstupní zprávy použijte proměnnou typu *string*. Textové řetězce se spojují operátorem +.



Příklad 3: Napište program, který zjistí, zda je zadaná hodnota celé číslo. Využijte konstrukci `try-catch` a převod zadaného textového řetězce na číslo. Pokud se povede, je to celé číslo.



7. Grafika

Napíšeme program, ve kterém vykreslíme do okna programu několik úseček a obrazců (nevyplněných).

Událost Paint

Abychom mohli kreslit do okna programu, musíme pro něj napsat obslužnou metodu události `Paint`.
Událost paint je pro okno programu vyvolána nejčastěji z následujících příčin:

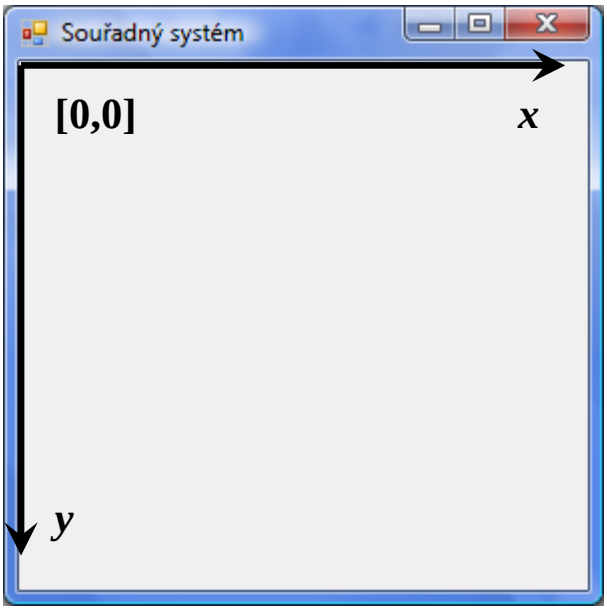
- Něž se okno porvé vykreslí (po spuštění programu)
- Když se okno minimalizuje na hlavní panel a poté obnoveno
- Jestliže je část okna něčím překryta a uživatel ji odkryje

Kreslící plocha

Obrazce musíme kreslit „někam“, tedy na nějakou kreslící plochu. Jako název pro kreslící plochu budeme používat název `kp`. Tuto proměnnou deklarujeme a zároveň inicializujeme příkazem

```
Graphicskp = e.Graphics;
```

Souřadný systém okna



Levý horní roh okna má souřadnice [0,0].*x*-ová souřadnice roste směrem doprava a *y*-ová roste směrem dolů (opačně než známe z matematiky). Hodnoty se udávají v pixelech, není-li uvedem jiná jednotka

Pera

Prvním parametrem metod pro kreslení obrazce je vždy *pero*. Pero je úplnou charakteristikou čáry, tj. kombinací barvy, tloušťky a stylu.
Pro začátek nám bude stačit tenká plná čára, kterou si můžeme vybrat ze „sady per“ `Pens`. Napíšeme `Pens.`*barva_čáry*. Např `Pens.Red`.

Grafické příkazy

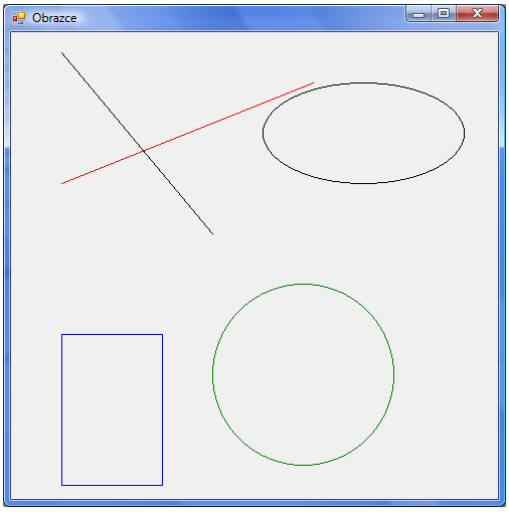
Obrazce se kreslí na kp, proto píšeme `kp.`*metoda_pro_kreslení_obrazce*(*parametry*);

Metoda	*Funkce*	*Parametry*
`DrawLine`	kreslí úsečku	pero, souřadnice koncových bodů
`DrawRectangle`	kreslí obdélník	pero, souřadnice levého horního rohu, šířka, výška
`DrawEllipse`	kreslí elipsu	pero, souř. levého hor. rohu opsaného obdélníka, šířka, výška

```
kp.DrawLine(Pens.Black,50,20,200,200);
```
Nakreslí černou úsečku z bodu [50,20] do bodu [200,200]

```
kp.DrawRectangle(Pens.Blue,50,300,100,150);
```
Nakreslí modrý obdélník s levým horním rohem [50,300], šířkou 100 a výškou 150. Čtverec je obdélník se stejnou šířkou a výškou.

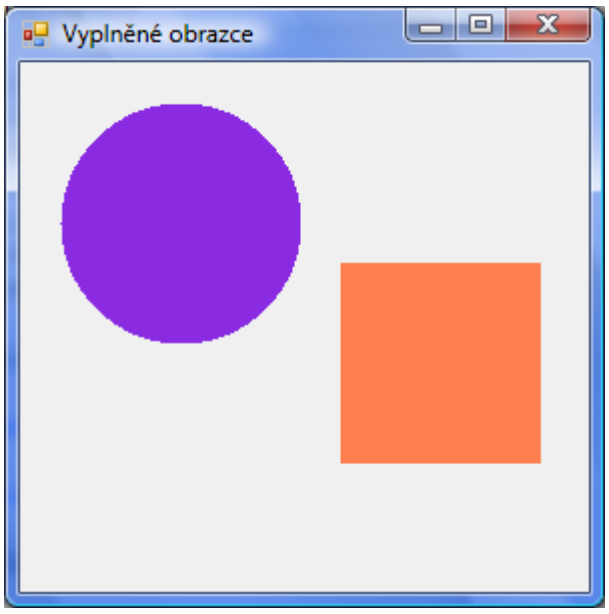
```
kp.DrawEllipse(Pens.Black,250,50,200,100);
```
Nakreslí elipsu s levým horním rohem opsaného obdélníka [250,50], šířkou 200 a výškou 100. Kružnice je elipsa se stejnou šírkou a výškou




```
privatevoid OknoProgramu_Paint(object sender,
PaintEventArgs e)
{
Graphics kp = e.Graphics;
  kp.DrawLine(Pens.Black,50,20,200,200);
  kp.DrawLine(Pens.Red,300,50,50,150);
  kp.DrawRectangle(Pens.Blue,50,300,100,150);
  kp.DrawEllipse(Pens.Black,250,50,200,100);
  kp.DrawEllipse(Pens.Green,200,250,180,180);
}
```

Vyplněné obrazce

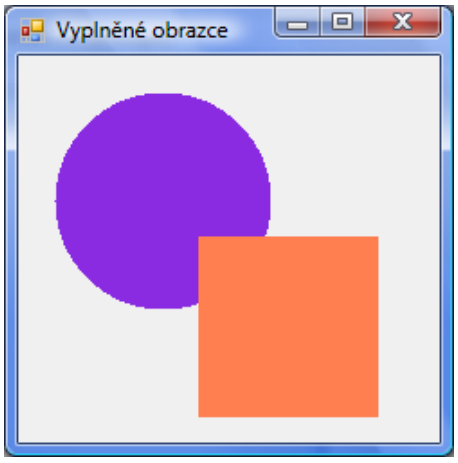
Kreslení vyplněných obrazců se od těch nevyplněných liší tím, že kreslíme štětcem (vybíráme ze „sady štětců“ `Brashes`) a názvy všech metod nezačínají `Draw`, ale `Fill`. Ostatní parametry jsou stejné



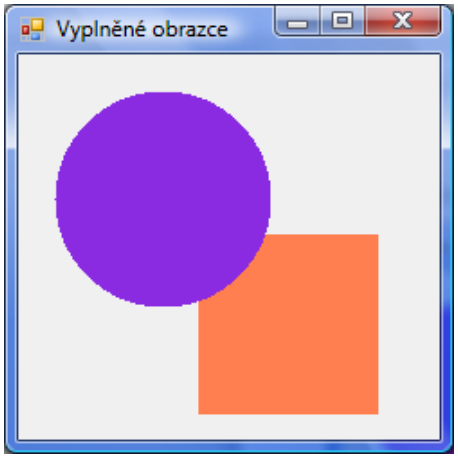
Př.

```
privatevoid OknoProgramu_Paint(object
sender, PaintEventArgs e)
{
Graphics kp = e.Graphics;
kp.FillEllipse(Brushes.BlueViolet,20,20,
120,120);
kp.FillRectangle(Brushes.Coral,160,100,1
00,100);
}
```

Při překrývání obrazců bude navrch ten, který je nakreslený později

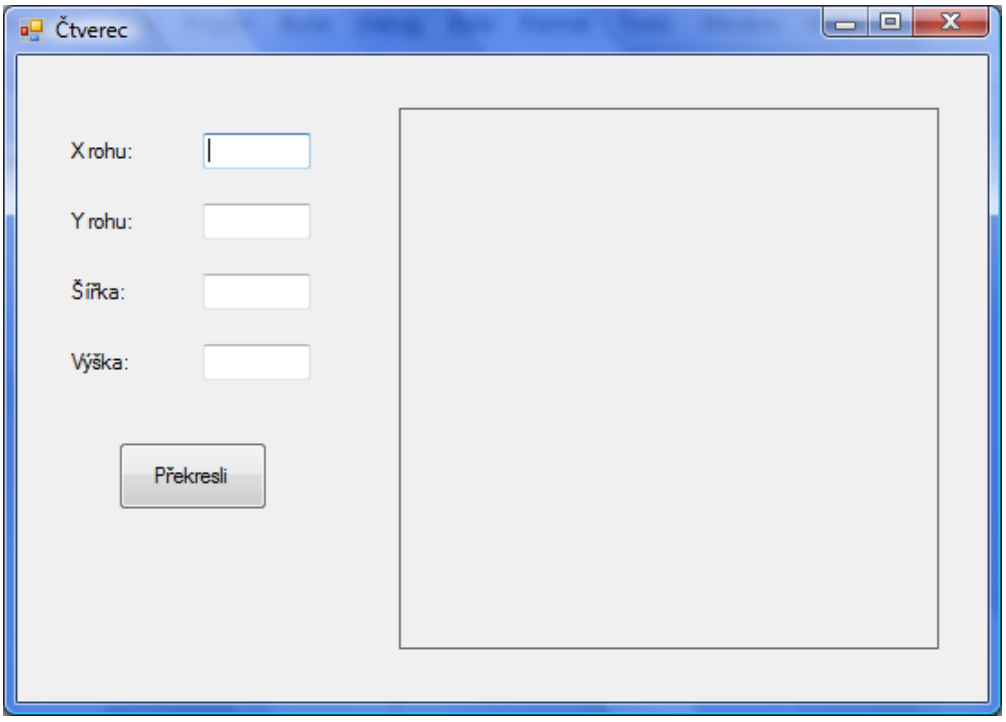



```
privatevoid OknoProgramu_Paint(object sender,
PaintEventArgs e)
{
Graphics kp = e.Graphics;
kp.FillEllipse(Brushes.BlueViolet,20,20,120,120);
kp.FillRectangle(Brushes.Coral,100,100,100,100);
}
```

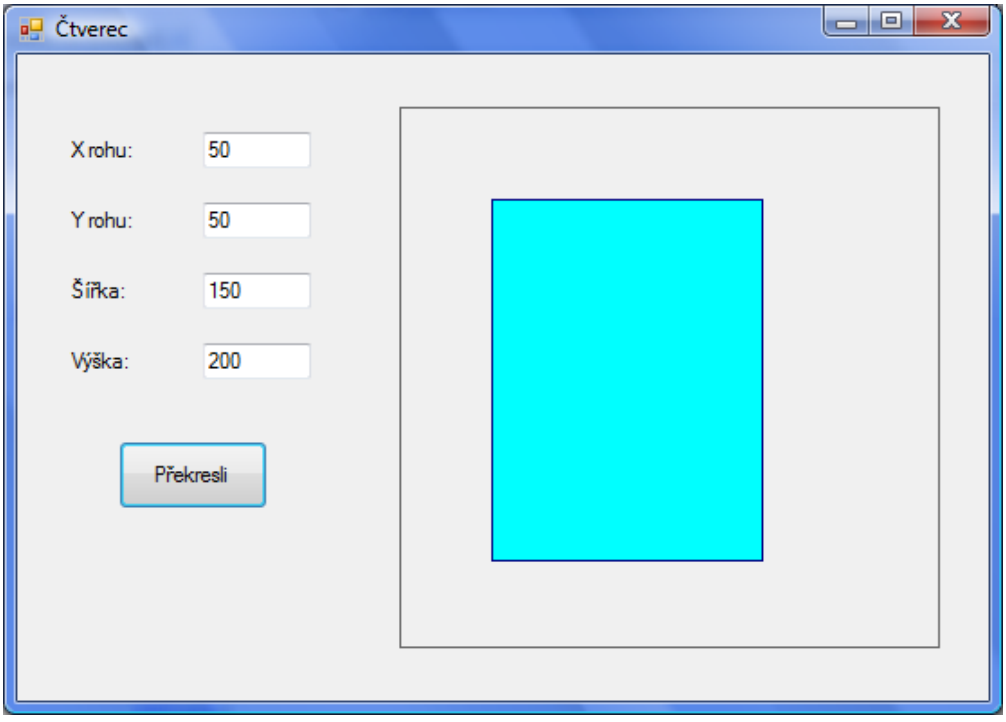



```
privatevoid OknoProgramu_Paint(object sender,
PaintEventArgs e)
{
Graphics kp = e.Graphics;
kp.FillRectangle(Brushes.Coral,100,100,100,100);
kp.FillEllipse(Brushes.BlueViolet,20,20,120,120);
}
```

Panel



Pokud nechceme kreslit přímo do okna programu, zvolíme v Toolboxu nástroj „*panel*“ ze skupiny *Containers*. Přetáhneme ho do okna programu a pro jeho ohraničení nastavíme vlastnost `BorderStyle` na hodnotu `FixedSingle`. Pro kreslení do panelu napíšeme metodu `panel_click`. Používání metod pro kreslení graf. objektů je stejné, souřadný systém se bere vzhledem k panelu, takže bod [0,0] je v levém horním rohu panelu.



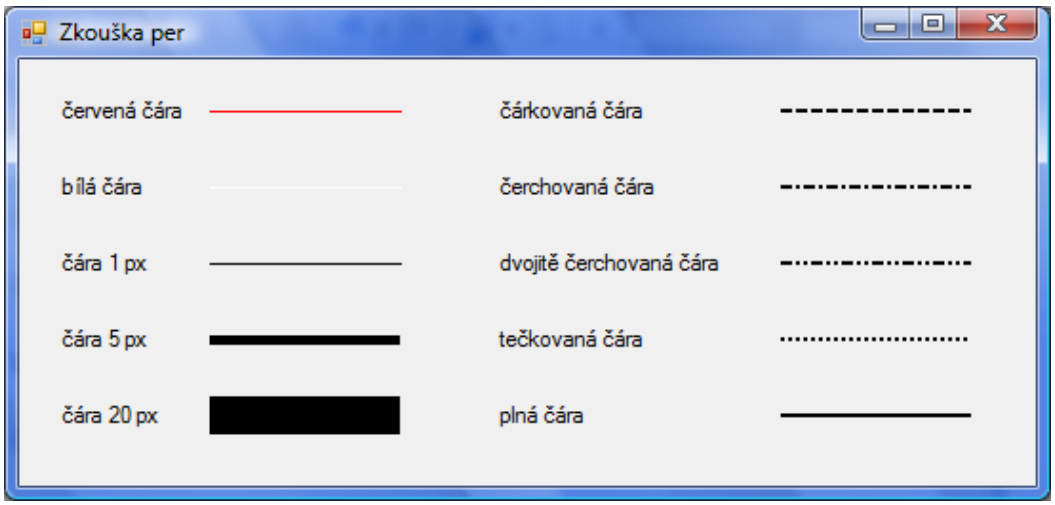
Připravíme si prostředí podle obrázku. Panelu nastavíme rozměry 300;300.
Po stisku tlačítka *Překresli* se do panelu vykreslí modrý (aqua) vyplněný obdélník s tmavě modrým okrajem (navy).

Souřadnice levého horního rohu, šířku a výšku vezmeme z textových polí. Hodnoty musíme převést z textového řetězce na celé číslo. Kvůli ošetření vstupních údajů napíšeme všechny převody do pokusného bloku `try`. Do větve catch napíšeme příkaz return; , který ukončí vykonávání obslužné metody.
Tentokrát nemůžeme použít deklaraci s inicilizací, protože proměnné, které bychom deklarovali v bloku try, by mimo tento blok neexistovaly a my bychom je nemohli využít pro kreslení obdélníku.
Proměnné stejného typu můžeme deklarovat jedním příkazem:`int x, y, šířka, výška;`
Nejprve nakreslíme plný obdélník a pak teprve jeho obrys. Obráceně by se nám obrys překryl výplní.
Překreslení panelu vyvoláme příkazem `Panel.Refresh();`, který napíšeme do metody `TlačítkoPřekresli_Click`

```
privatevoid Panel_Paint(object sender, PaintEventArgs e)
{
     Graphics kp = e.Graphics;
     int x, y, šířka, výška;
     try
        {
           x = Convert.ToInt32(PoleX.Text);
           y = Convert.ToInt32(PoleY.Text);
           šířka = Convert.ToInt32(PoleŠířka.Text);
            výška = Convert.ToInt32(PoleVýška.Text);
     }
     catch {return;}
     kp.FillRectangle(Brushes.Aqua, x, y, šířka, výška);
     kp.DrawRectangle(Pens.Navy,x,y,šířka,výška);
}

privatevoid TlačítkoPřekresli_Click(object sender, EventArgs e)
 {
     Panel.Refresh();
 }
```

Různá pera

Pokud nám nestačí tenká plná čára ze sady per `Pens`, můžeme si vyrobit vlastní pero, jako objekt třídy `Pen`.
Pro vytvoření pera červené barvy napíšeme: `Pen MojePero=newPen(Color.Red);`
Jako druhý parametr můžeme uvést tloušťku pera. Není-li tloušťka uvedena, bere se 1px.
Barvu a tloušťku pera můžeme měnit zadáním hodnot do `MojePero.ColoraMojePero.Width.`
Pro definování stylu čáry musíme používat jmenný prostor `System.Drawing.Drawing2Da` na začátek souboru Form1.cs vložit řádek `using System.Drawing.Drawing2D;`
Styl čáry zadáme hodnotou do `MojePero.DashStyle`.

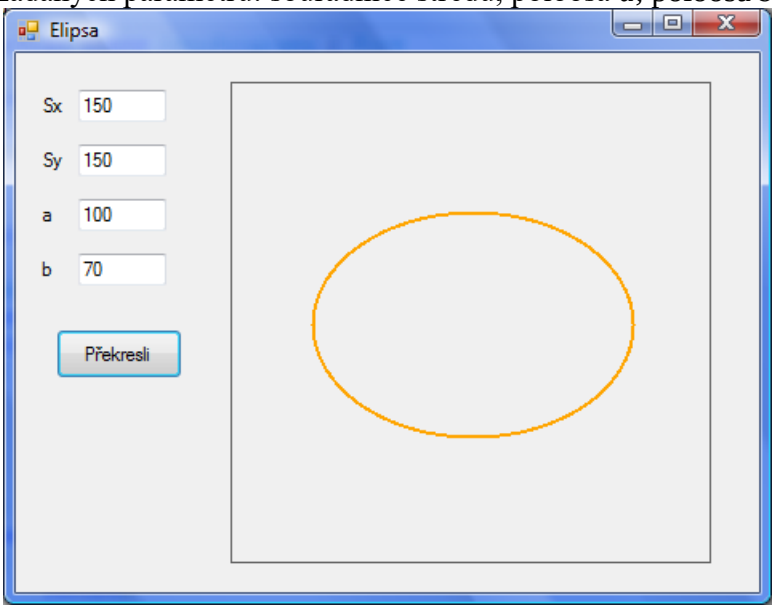


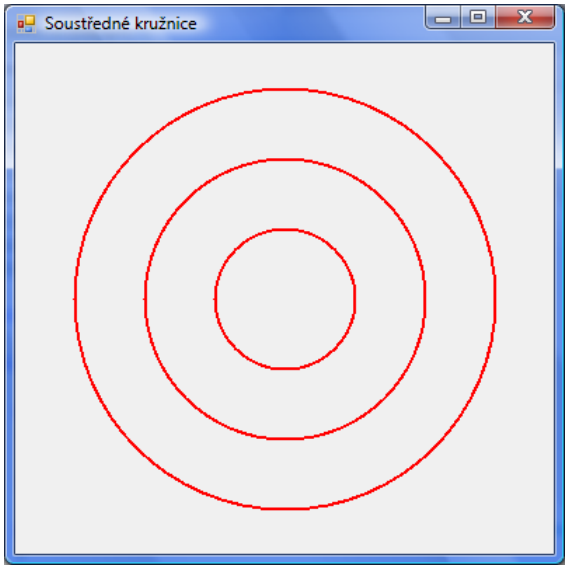
Hodnoty jsou:
Dash – čárkovaná
Dot - tečkovaná
DashDot – čerchovaná (čárka-tečka)
DashDotDot – dvojitě čerchovaná (čárka-čárka-tečka)
Solid – plná

```
privatevoid OknoProgramu_Paint(object sender, PaintEventArgs e)
{
Graphics kp = e.Graphics;
Pen MojePero=newPen(Color.Red);
            kp.DrawLine(MojePero, 100,27,200,27);
            MojePero.Color = Color.White;
            kp.DrawLine(MojePero, 100, 67, 200, 67);
            MojePero.Color = Color.Black;
            MojePero.Width = 1;
            kp.DrawLine(MojePero, 100, 107, 200, 107);
            MojePero.Width = 5;
            kp.DrawLine(MojePero, 100, 147, 200, 147);
            MojePero.Width = 20;
            kp.DrawLine(MojePero, 100, 187, 200, 187);
            MojePero.Width = 2;
            MojePero.DashStyle = DashStyle.Dash;
            kp.DrawLine(MojePero, 400, 27, 500, 27);
            MojePero.DashStyle = DashStyle.DashDot;
            kp.DrawLine(MojePero, 400, 67, 500, 67);
            MojePero.DashStyle = DashStyle.DashDotDot;
            kp.DrawLine(MojePero, 400, 107, 500, 107);
            MojePero.DashStyle = DashStyle.Dot;
            kp.DrawLine(MojePero, 400, 147, 500, 147);
            MojePero.DashStyle = DashStyle.Solid;
            kp.DrawLine(MojePero, 400, 187, 500, 187);
    }
```

Příklady na procvičení:

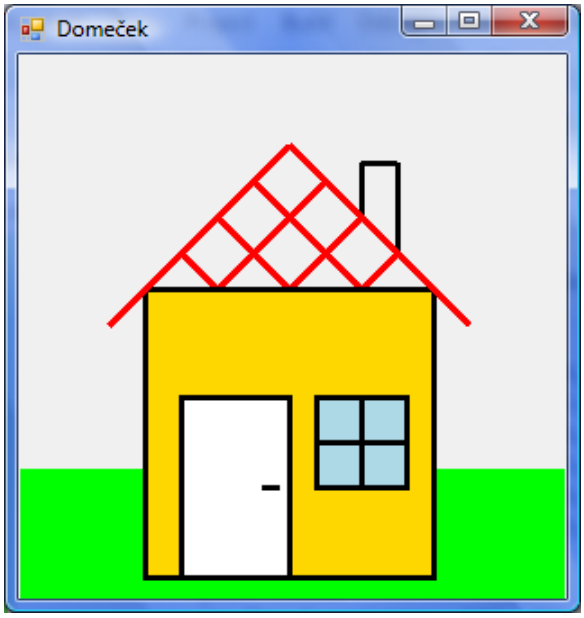
Příklad 1: Napište program*Elipsa*, který do panelu nakreslí oranžovou elipsu s tloušťkou čáry 2px, podle zadaných parametrů: souřadnice středu, poloosa *a*, poloosa *b*.

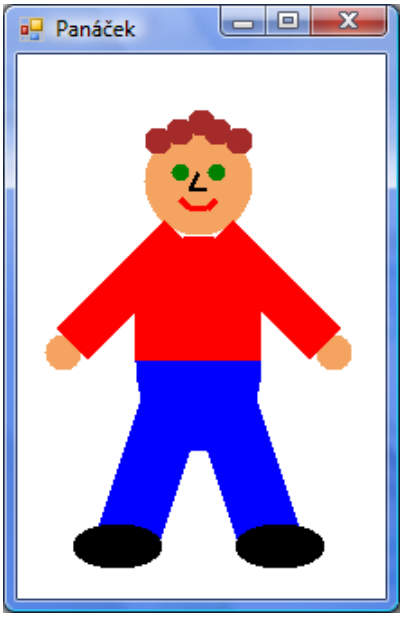


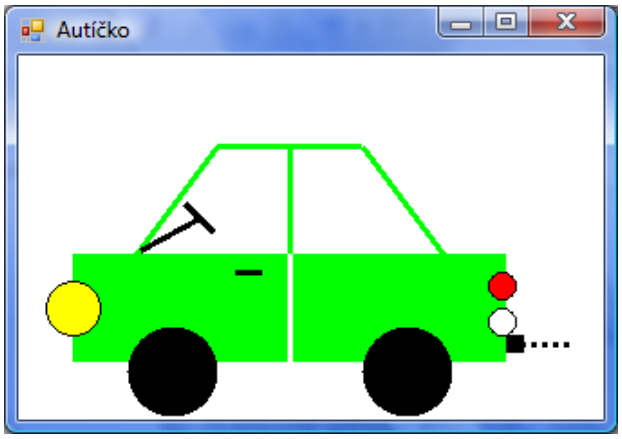


Příklad 2: Napište program *Soustředné kružnice*, který nakreslí do okna programu tři červené kružnice s tloušťkou čáry 2px, se středem v středu okna a poloměry 50, 100 a 150px.
Střed klientské části okna zjistíte: `int Sx = ClientSize.Width / 2;`
`int Sy = ClientSize.Height / 2;`

Příklad 3: Nakreslete do okna programu některý z těchto obrázků: domeček, panáček, autíčko.



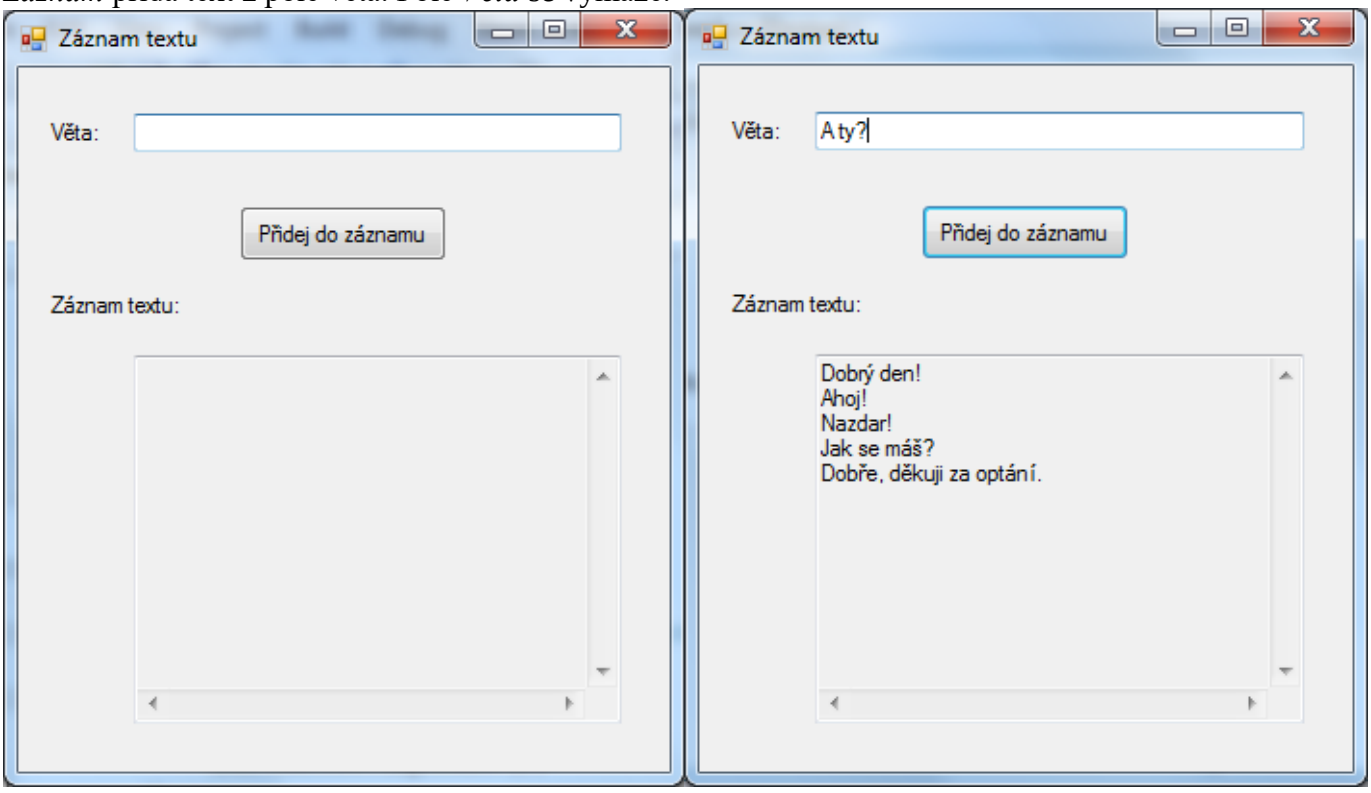




8. Práce s proměnnými

Složené přiřazení

Připravíme si prostředí pro nový projekt *Záznam textu.* Po stisknutí tlačítka *Přidej do záznamu* se do pole *Záznam* přidá text z pole věta. Pole *Věta* se vymaže.



Pro pole *Záznam* je vhodné nastavit následující vlastnosti:

Vlastnost	*hodnota*	
`Multiline`	`True`	pole půjde roztáhnout do více řádků
`ReadOnly`	`True`	pouze pro čtení – uživatel nemůže do pole zapisovat
`ScrollBars`	`Both`	zobrazí se oba posuvníky
`WordWrap`	`False`	text se nebude zalamovat – po stisku tlačítka vznikne jeden řádek

Pro přidání *Věty* do *Záznamu* je třeba k dosavadnímu obsahu pole *Záznam* připojit obsah pole *Věta*.

```
PoleZáznam.Text = PoleZáznam.Text + PoleVěta.Text;
```

Pro zjednodušení zápisu lze použít **Složené přiřazení** pomocí operátoru += .

```
PoleZáznam.Text += PoleVěta.Text;
```

Pro odřádkování upravíme přiřazení do konečné podoby

```
PoleZáznam.Text += PoleVěta.Text + Environment.NewLine;
```

Celý zápis metody:

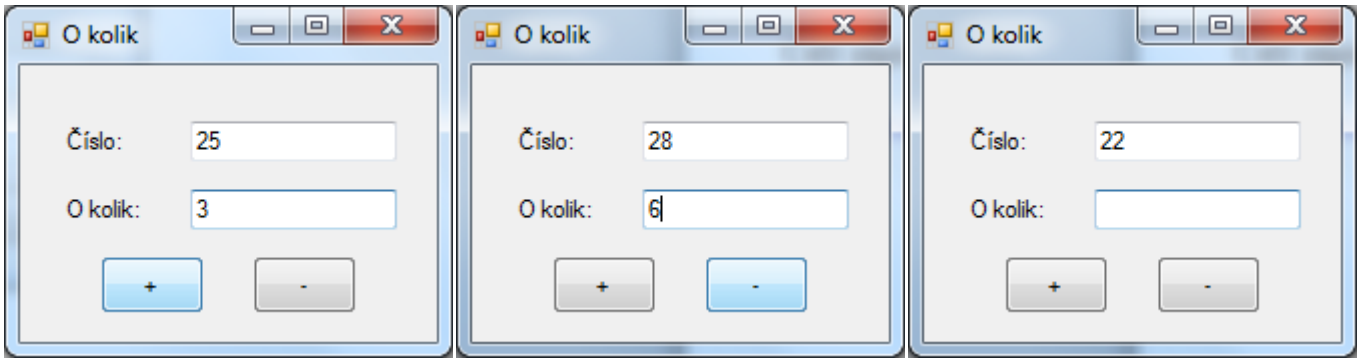
```
privatevoid TlačítkoPřidej_Click(object sender, EventArgs e)
        {
            PoleZáznam.Text += PoleVěta.Text + Environment.NewLine;
            PoleVěta.Text = null;
        }
```

U řetězců používáme pro spojování operátor +=. U celých čísel lze použít i operátory += a -=.
Za operátorem uvádíme o kolik se zvětší nebo zmenší. Pro zvětšení nebo zmenšení o 1 lze též použít ++ a --.

Př. `int číslo = 16;`

číslo	číslo+=2	číslo-=5	číslo++	číslo--
16	18	11	17	15

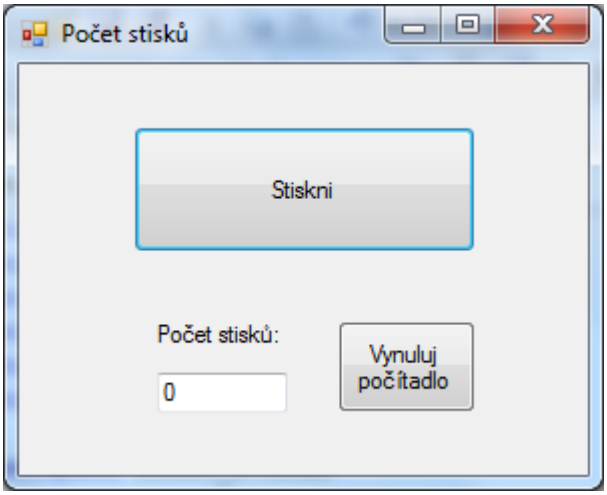
Př.




```
privatevoid TlačítkoPlus_Click(object sender, EventArgs e)
  {
     int číslo = Convert.ToInt32(PoleČíslo.Text);
     int kolik = Convert.ToInt32(PoleKolik.Text);
     číslo += kolik;
     PoleČíslo.Text = Convert.ToString(číslo);
}

privatevoid TlačítkoMínus_Click(object sender, EventArgs e)
  {
int číslo = Convert.ToInt32(PoleČíslo.Text);
int kolik = Convert.ToInt32(PoleKolik.Text);
     číslo -= kolik;
     PoleČíslo.Text = Convert.ToString(číslo);
}
```

Členské proměnné

Podle obrázku vytvoříme nový projekt.



Zápis metod pro obě tlačítka je jednoduchý:

```
privatevoid tlačítkoStiskni_Click(object
sender, EventArgs e)
{
 počet++;
 PolePočet.Text =
 Convert.ToString(počet);
}

privatevoid TlačítkoVynuluj_Click(object
sender, EventArgs e)
{
 počet = 0;
 PolePočet.Text =
 Convert.ToString(počet);
   }
```

Složitější bude vymyslet, kde deklarovat a inicalizovat proměnnou *počet*. Proměnné, které jsme dosud používali, byly deklarované vždy v nějaké metodě a existovaly pouze po dobu existence (vykonávání) dané metody. Takto definované proměnné se nazývají **lokální proměnné**. Lokální proměnná nám v tomto případě nepomůže. Po vykonání metody`tlačítkoStiskni_Click` by proměnná zanikla a při dalším stisknutí bychom počítali znovu od nuly. Navíc bychom nemohli tuto proměnnou použít v metodě `TlačítkoVynuluj_Click`.
Dalo by se to řešit jedině tak, že bychom si opakovaně načítali aktuální počet z pole *Počet*. Vhodnější bude použít takovou proměnnou, která bude existovat po celou dobu běhu programu, tj.po dobu existence *Okna Programu*. Taková proměnná se deklaruje jako složka třídy `OknoProgramu`. Tento typ proměnné nazýváme **členská proměnná**.

Výpis celého zdrojového kódu:

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace Počet_stisků
{
Public partial classOknoProgramu : Form
{
int počet = 0;
public OknoProgramu()
        {
            InitializeComponent();
        }
privatevoid tlačítkoStiskni_Click(object sender, EventArgs e)
        {
            počet++;
            PolePočet.Text = Convert.ToString(počet);
        }
privatevoid TlačítkoVynuluj_Click(object sender, EventArgs e)
        {
            počet = 0;
            PolePočet.Text = Convert.ToString(počet);
        }
    }
}
```

Pohyblivý obrázek

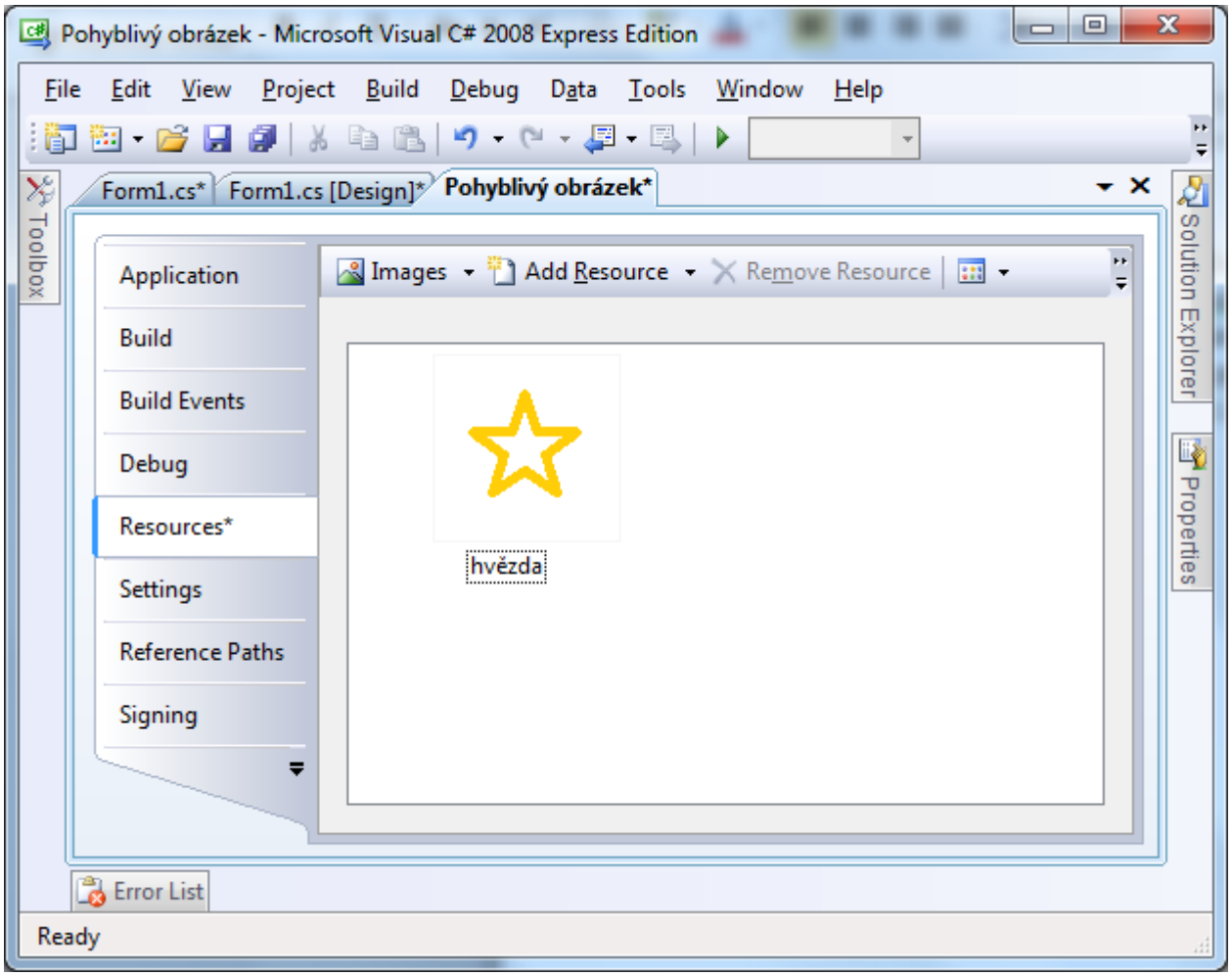
Pro následující projekt si nejprve vytvoříme obrázek velikosti s bílým nebo transparentním pozadím. Pro vytvoření můžeme použít např. program Malování. Nastavíme si velikost obrázku a vložíme některý z automatických tvarů. Já budu používat obrázek *hvězda.png* s velikostí 60x60 px a bílým pozadím. Obrázek budeme kreslit na panel pomocí metody `DrawImage`, jejímiž parametry jsou v daném pořadí: obrázek, x-ová souřadnice levého horního rohu, y-ová souřadnice levého horního rohu, šířka obrázku, výška obrázku.

Obrázek vložíme do složky s `.exe` souborem, tj. do složky `bin\debug`. Pro přiřazení správného obrázku použijeme příkaz: `Image obrázek = Image.FromFile("hvězda.png");`
Tento způsob zvolíme, pokud chceme mít možnost obrázek měnit. Nesmíme ho ovšem z dané složky smazat.

Druhou možností je použít vestavěný obrázek, který bude přímo součástí `.exe` souboru. Postup je následující: - z nabídky *Project* vybereme *„název projektu“ Properties*. (v našem případě *Pohyblivý obrázek Properties*).

- v levé části okna, které se objeví, zvolíme *Resources*
- v nabídce klikneme na šipku vedle položky *AddResource*.
- v podnabídce, která se rozbalí zvolíme*AddExistingFile*
- v dialogovém okně najdeme obrázek a dáme *Open*

Tímto jsme obrázek zařadili mezi vestavěná data projektu. Příkaz pro přiřazení správného obrázku bude vypadat takto: `Image obrázek = Properties.Resources.hvězda;`



Pro souřadnice a velikost panáčka vytvoříme čtyři proměnné typu *integer*. Velikost nastavíme napevno 60x60. Levý horní roh určíme tak, aby byl obrázek přibližně v prostředku. Musíme uvažovat velikost panelu a velikost obrázku. V našem případě je panel 300x300 a obrázek 60x60. Zvolíme tedy levý horní roh [120;120].
Všechny proměnné (*obrázek, x, y, šířka, výška*) deklarujeme jako členské.

Metody pro obsluhu tlačítek pak budou jednoduché. V každé změníme jednu ze souřadnic (vpravo a vlevo *x*, nahoru a dolů *y*) o 5px (vlevo a nahoru -5, vpravo a dolů +5). Po změně souřadnice znovu vykreslíme panel příkazem `panel.Refresh();`
Celý zápis zdrojového kódu:

```
Public partial classOknoProgramu : Form
{
      int x = 120;
      int y = 120;
      int výška = 60;
      int šířka = 60;
      Image obrázek = Properties.Resources.hvězda;

public OknoProgramu()
        {
            InitializeComponent();
        }

privatevoid panel_Paint(object sender, PaintEventArgs e)
        {
            Graphics kp = e.Graphics;
            kp.DrawImage(obrázek,x,y,šířka,výška);
        }

privatevoid TlačítkoNahoru_Click(object sender, EventArgs e)
        {
            y -= 5;
            panel.Refresh();
        }

privatevoid TlačítkoDolů_Click(object sender, EventArgs e)
        {
            y += 5;
            panel.Refresh();
        }

privatevoid TlačítkoDoleva_Click(object sender, EventArgs e)
        {
            x -= 5;
            panel.Refresh();
        }

privatevoid TlačítkoDoprava_Click(object sender, EventArgs e)
        {
            x += 5;
            panel.Refresh();
        }
}
```

9. Čísla

Desetinná čísla

Dosud jsme pracovali pouze s celými čísly. Pokud budeme chtít používat výpočty s desetinnými čísly, budeme místo typu int používat typ `double`. Metoda pro převod z řetězce pak bude `Convert.ToDouble()`.

Desetinná čísla zadáváme do zdrojového kódu s desetinnou tečkou (nikoliv čárkou). Uživatel do textového pole pro zápis ovšem používá desetinnou čárku. Výpočty s desetinnými čísly jsou náročnější a musíme také počítat se zaokrouhlovací chybou.
Pro matematické operace **sčítání**, **odčítání**, **násobení** a **dělení** používáme operátory +, -, * a /.
Priorita operací je stejná jako v matematice. Závorky pro změnu proirity pouze kulaté.
Pro druhou nebo i třetí **mocninu** použijeme násobení. Pro jiné mocniny použijeme metodu `Math.Pow(základ, exponent)`.
Pro druhou **odmocninu** použijeme metodu `Math.Sqrt(argument odmocniny)`. Ostatní odmocniny počítáme jako mocniny s racionálním exponentem ($\sqrt[n]{x^m} = x^{\frac{m}{n}}$).
Pro goniometrické fce **sinus**, **kosinus** a **tangens** máme metody `Math.Sin(úhel)`, `Math.Cos(úhel)` a `Math.Tan(úhel)`. Úhel uvádíme v radiánech. Pro převod ze stupňů na radiány použijeme vztah `úhelVRad = úhelVeStupních / 180 * Math.Pi;`
Inverzní fce jsou `Math.Asin(argument)`, `Math.Acos(argument)` a `Math.Atan(argument)`.
Absolutní hodnota je `Math.Abs(argument)`. **Logaritmus** je `Math.Log(argument,základ)`,

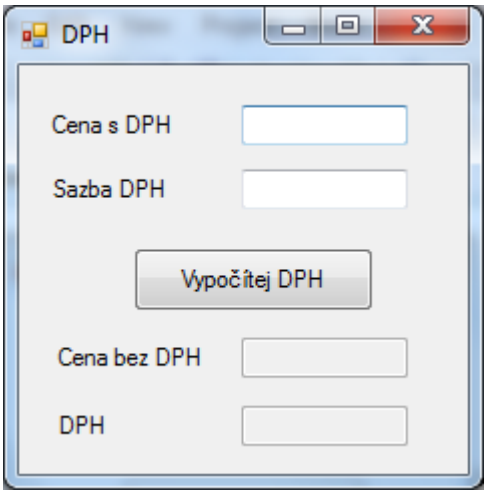
Příklady výpočtů:

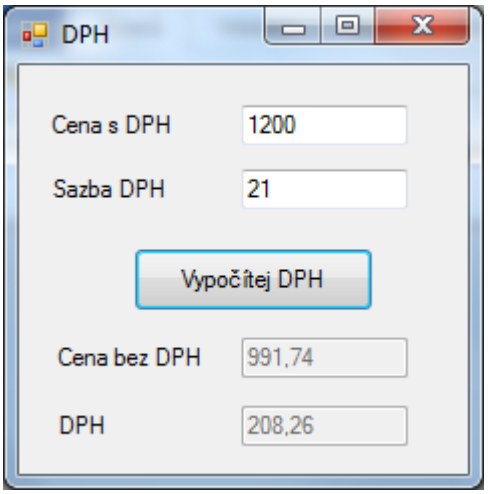
$\frac{x+1}{2x-5}$	`(x+1)/(2*x-5)`
$-5x^2+2(x-0{,}5)$	`-5*x*x+2*(x-0.5)`
2^x	`Math.Pow(2,x)`
$\sqrt{\frac{x}{5}}$	`Math.Sqrt(x/5)`
$\sqrt[5]{x-1}$	`Math.Pow(x-1,1.0/5.0)` nebo `Math.Pow(x-1,0.2)`
$\log_2\lvert x+1\rvert$	`Math.Log(Math.Abs(x+1),2)`
$\sin x$	`Math.Sin(x)`

Př.

```
privatevoid TlačítkoVypočítej_Click(object sender, EventArgs e)
{
double cenaS, Sazba;
try
    {
        cenaS = Convert.ToDouble(PoleCenaS.Text);
        Sazba = Convert.ToDouble(PoleSazba.Text);
    }
catch
    {
      MessageBox.Show("chybně zadaná vstupní data");
      return;
    }
double cenaBez = cenaS / (1 + Sazba / 100);
double DPH = cenaS - cenaBez;
PoleCenaBez.Text = cenaBez.ToString("F2");
   PoleDph.Text = DPH.ToString("F2");
}
```

V příkladu jsme použili jiný způsob převodu čísla na řetězec. Řetězec `"F2"` v závorce znamená výpis čísla na dvě desetinná místa. Tento zápis by byl možný i u čísla typu `int`.



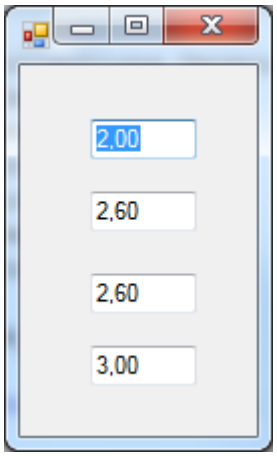


Celočíselné dělení

Při použití operátoru / musíme vědět, že dělení dvou celých čísel je celočíselné dělení. Výsledek celočíselného dělení např. `17/5` je `3`, a to i v případě, že výsledek dělení přiřazujeme proměnné typu `double`.

Pokud chceme získat desetinné číslo, musí být alespoň jedno z čísel desetinné, tj. `17.0/5` nebo `17/5.0` nebo `17.0/5.0`

Při použití celočíselného dělení se nám někdy může hodit i operace *zbytek po dělení*, pro kterou se používá operátor %.




```
privatevoid Form1_Load(object sender, EventArgs e)
{
      Double x = 13 / 5;
      pole1.Text = x.ToString("F2");
      x = 13.0 / 5;
      pole2.Text = x.ToString("F2");
      x = 13 / 5.0;
      pole3.Text = x.ToString("F2");
      x = 13 % 5;
      pole4.Text = x.ToString("F2");
}
```

Typová konverze

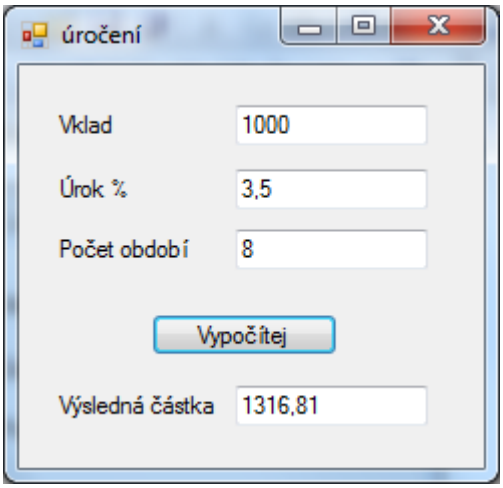
Kromě typů int a double existuje řada dalších číselných typů, které se liší rozsahem hodnot. Jejich znalost pro nás není zatím důležitá a vystačíme s těmito dvěma typy. Co bychom, ale vědět měli je, jak můžeme mezi těmito dvěma typy vzájemně přiřazovat.

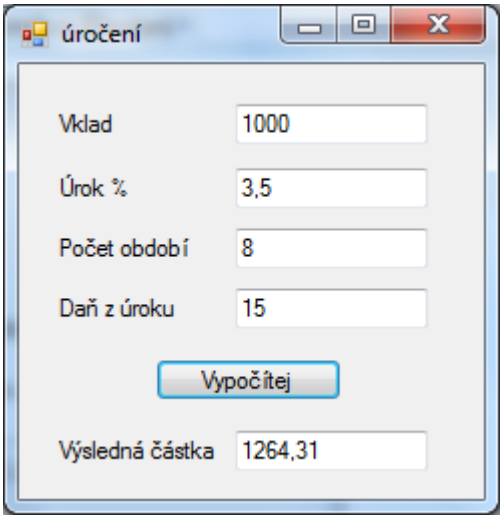
Hodnoty typu int můžeme do typu double vkládat libovolně. Obráceně to nelze. Pokud do typu int chceme vložit číslo double, musíme ho nejprve převést na celé číslo tím, že před číslo typu double napíšeme do závorky (int).

Př.
```
double x=5.2;
int y=2;
x=y;
y=(int) x;
```

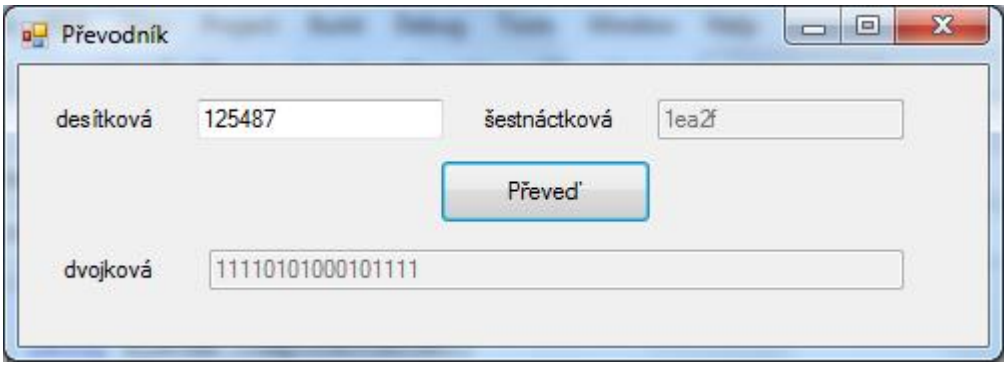
Příklady na procvičení

1. Napište program úročení, který po zadání vkladu, úroku a počtu úrokovacích období vypočítá výslednou částku. Řešení můžete vylepšit přidáním daně z úroku.



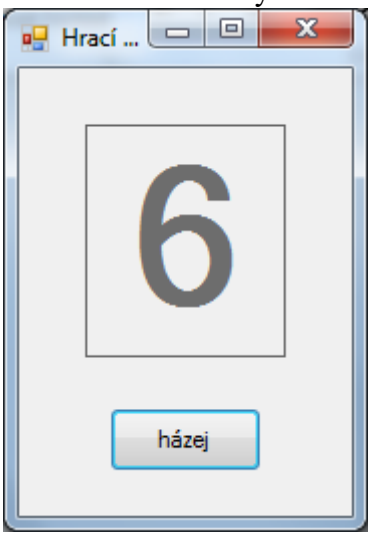


2. Napište převodník do dvojkové a šestnáctkové soustavy. Využijte metodu `Convert.ToString` se dvěma parametry – první je převáděné číslo a druhý je základ soustavy, do které chcete číslo převést, tedy 2 nebo 16.



Náhodná čísla

Generování náhodných čísel si ukážeme na programu, který simuluje hrací kostku.




```
public partial class oknoProgramu : Form
{
Random náhoda = new Random();
public oknoProgramu()
 {
   InitializeComponent();
 }

privatevoid tlačítkoHázej_Click(object sender,
EventArgs e)
{
int číslo = náhoda.Next(1, 7);
  poleČíslo.Text = číslo.ToString();
 }
}
```

Deklaraci `náhody` musíme provést v místě, kde se deklarují členské proměnné. Paramatry metody `náhoda.Next(1, 7)` udávají dolní a horní mez generovaných čísel. Horní mez se uvádí o 1 větší.

10. Podmínka, větvení

Zatím jsme metody psali tak, že se vykonaly všechny příkazy, navíc v zadaném pořadí (kromě použití bloku `try-catch` na ošetření běhové chyby). Někdy potřebujeme vykonání příkazu provést jen za určitých podmínek.

Neúplná podmínka – řídicí konstrukce `if`

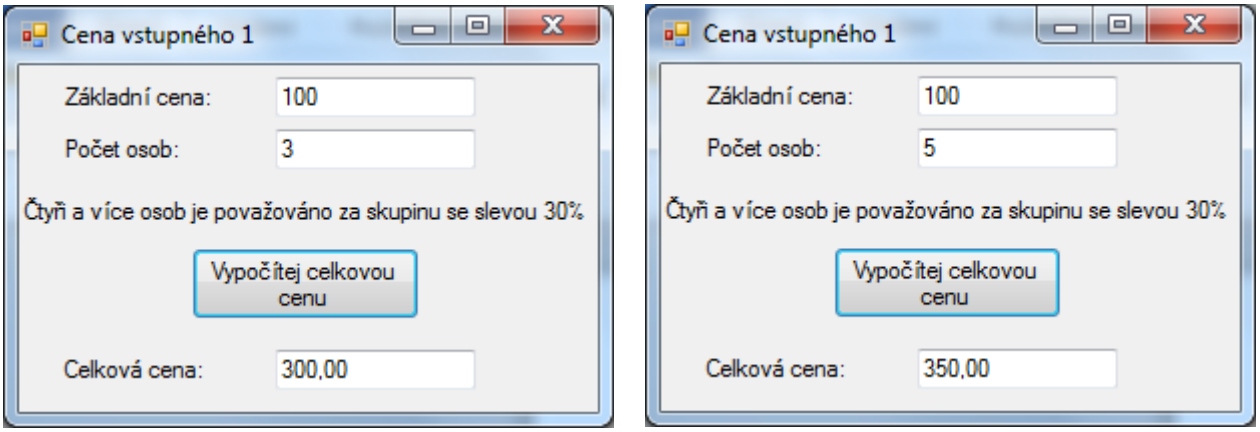
K zápisu podmínky slouží řídící konstrukce `if`.

```
if (podmínka)          nebo        if (podmínka)
    příkaz;                        {příkaz1;
                                    příkaz2;
                                    …
                                   }
```

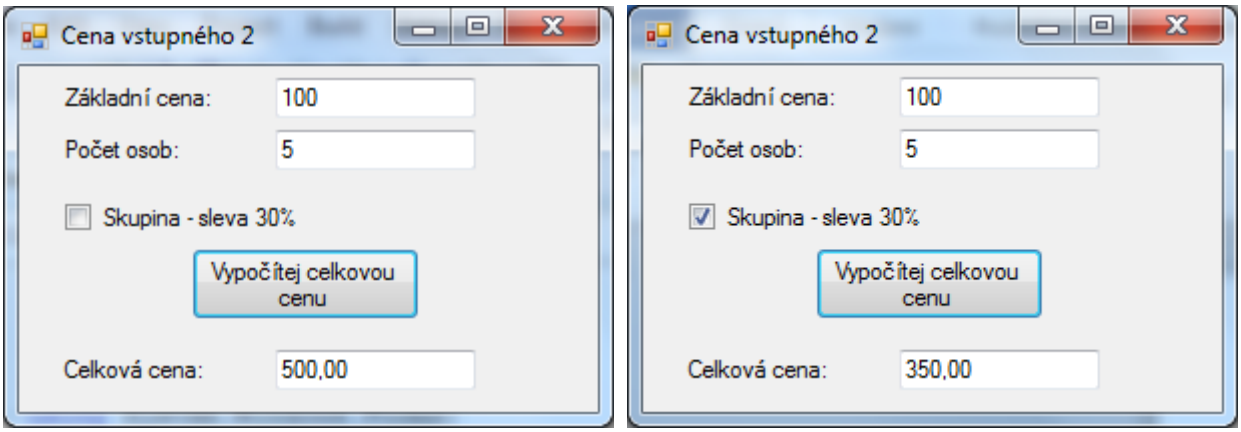
Je-li podmínka splněna, vykoná se příkaz(y). Není-li splněna, nestane se nic.
Podmínkou se myslí logický výraz, tj. výraz, který nabývá hodnoty `True` nebo `False`. Může to být vlastnost objektu nebo proměnná, která je logického typu (např. vlastosti `Checked` nebo `Enabled`). Nebo je to výsledek porovnávání pomocí relačního operátoru.

Relační operátory:		
	$a = b$	`a==b`
	$a \neq b$	`a!=b`
	$a < b$	`a<b`
	$a \leq b$	`a<=b`
	$a > b$	`a>b`
	$a \geq b$	`a>=b`

Př.



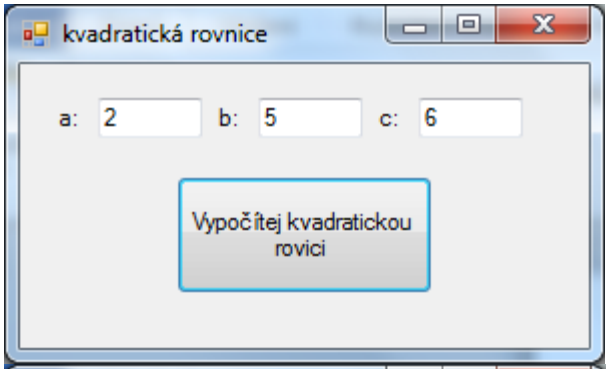

```
privatevoid TlačítkoVypočítej_Click(object sender, EventArgs e)
{
double počet, cena;
try
   {
   počet = Convert.ToDouble(PolePočetOsob.Text);
        cena = Convert.ToDouble(PoleZákladníCena.Text);
}
catch { return; }
if (počet >= 4)
   cena = 0.7 * cena;
double celkováCena = počet * cena;
   PoleCelkem.Text = celkováCena.ToString("F2");
 }
```

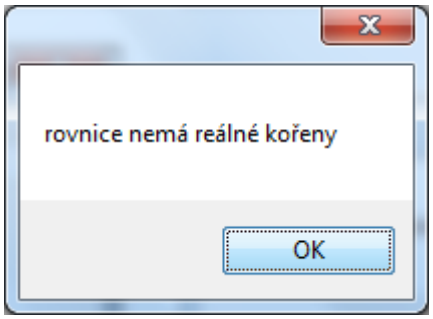


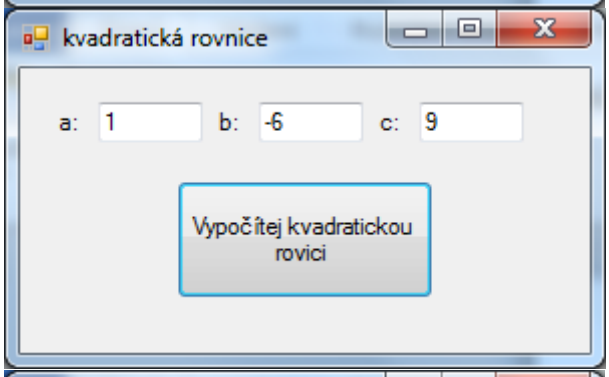

```
privatevoid TlačítkoVypočítej_Click(object sender, EventArgs e)
{
double počet, cena;
try
     {
          počet = Convert.ToDouble(PolePočetOsob.Text);
          cena = Convert.ToDouble(PoleZákladníCena.Text);
     }
catch { return; }
if (PolíčkoSkupina.Checked)
          cena = 0.7 * cena;
double celkováCena = počet * cena;
   PoleCelkem.Text = celkováCena.ToString("F2");
}
```

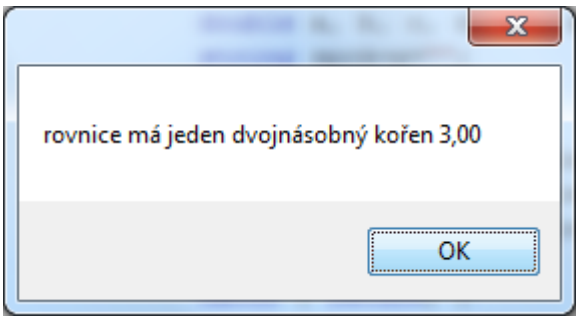
Př. **Kvadratická rovnice**

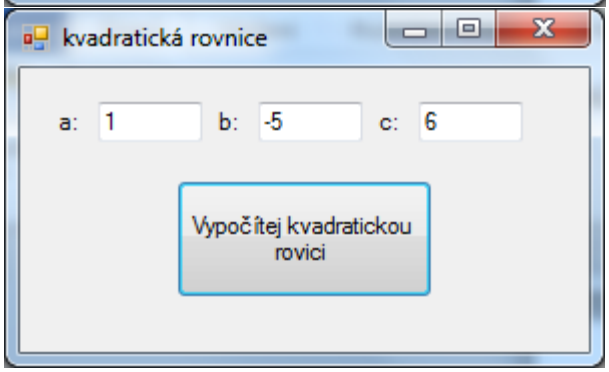
```
privatevoid TlačítkoVypočítej_Click(object sender, EventArgs e)
{
double a, b, c, d, x1, x2;
string zpráva="";
try
     {
       a = Convert.ToDouble(PoleA.Text);
       b = Convert.ToDouble(PoleB.Text);
       c = Convert.ToDouble(PoleC.Text);
     }
catch { return; }
d = b * b - 4 * a * c;
if (d > 0)
     {
        x1 = (-b + Math.Sqrt(d)) / (2 * a);
        x2 = (-b - Math.Sqrt(d)) / (2 * a);
zpráva="rovnice má dva kořeny "+x1.ToString("F2")+"a"+x2.ToString("F2");
     }
if (d == 0)
     {
       x1 = (-b) / (2 * a);
      zpráva = "rovnice má jeden dvojnásobný kořen " + x1.ToString("F2");
     }
if (d < 0)
zpráva = "rovnice nemá reálné kořeny";
MessageBox.Show(zpráva);
}
```

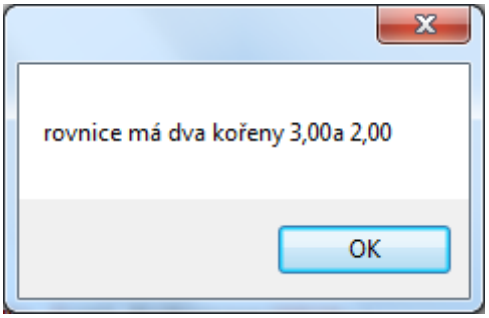












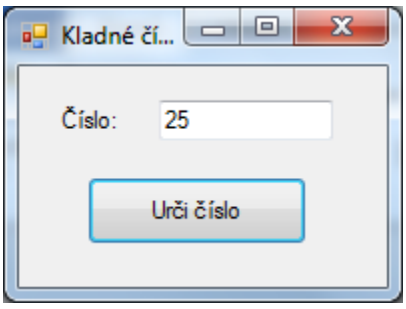
Úplná podmínka – řídicí konstrukce `if - else`

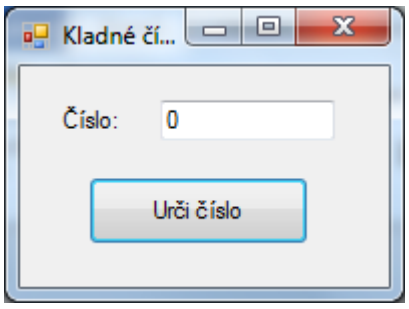
K zápisu podmínky slouží řídící konstrukce `if - else`.

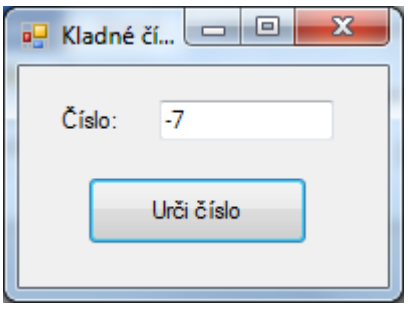
```
if (podmínka)
{blok 1}
else
     {blok 2}
```

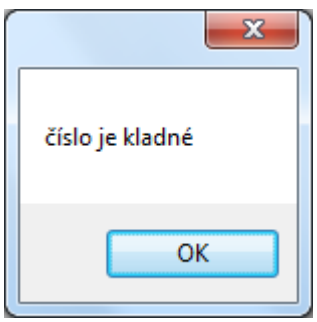
Je-li podmínka splněna, vykoná se příkaz(y) z bloku 1, Není-li splněna, vykoná se příkaz(y) z bloku 2. Pokud blok obsahuje pouze jeden příkaz, nemusí být složené závorky.

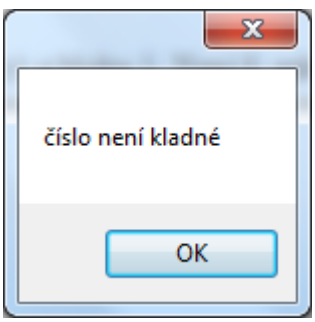
Př.











```
privatevoid Tlačítko_Click(object sender, EventArgs e)
{
int číslo;
string zpráva="";
try
{ číslo = Convert.ToInt32(PoleČíslo.Text); }
catch {return;}
if (číslo>0)
zpráva="číslo je kladné";
else
zpráva="číslo není kladné";
MessageBox.Show(zpráva);
}
```

Vnořené větvení

Pomocí vnořování konstrukcí `if – else` se dá program větvit na více než dvě větve. Zápis pak ovšem může být dost nepřehledný.

Upravená část kódu příkladu *kladné číslo*, tak aby bylo možno rozlišit i nulu a záporné číslo:

```
if (číslo>0)
        zpráva="číslo je kladné";
else if (číslo<0)
        zpráva="číslo je záporné";
     else
        zpráva="číslo je nula";
```

Upravená část kódu příkladu *kvadratickárovnice*

```
if (d > 0)
{
   x1 = (-b + Math.Sqrt(d)) / (2 * a);
x2 = (-b - Math.Sqrt(d)) / (2 * a);
zpráva="rovnice má dva kořeny "+x1.ToString("F2")+"a"+x2.ToString("F2");
}
else if (d == 0)
{
 x1 = (-b) / (2 * a);
zpráva = "rovnice má jeden dvojnásobný kořen " + x1.ToString("F2");
}
else zpráva = "rovnice nemá reálné kořeny";
```

Násobné větvení – řídicí konstrukce `switch`

Program můžeme větvit také na základě konkrétních hodnot určité proměnné.

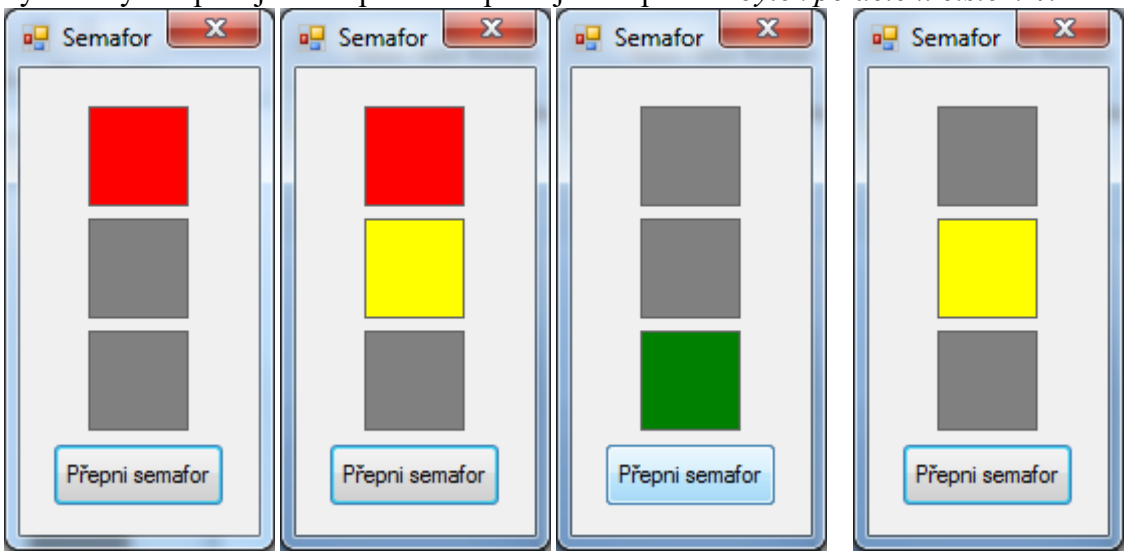
```
switch (proměnná)
{
  case hodnota1:
    příkazy;
    break;
  case hodnota2:
    příkazy;
    break;
   …
}
```

Užití si ukážeme na následujícím příkladu

Př. Vytvořte projekt *Semafor*. Pro světla semaforu použijte panelya zadávejte barvu jejich pozadí (`BackColor`) . Tlačítkem přepínejte jako na skutečném semaforu .

Semafor má čtyři stavy
- 0 - svítí pouze červená
- 1 - svítí červená a žlutá
- 2 - svítí zelená
- 3 - svítí žlutá

Tyto stavy se opakují. Pro opakování použijeme operaci *zbytek po dělení číslem 4*.



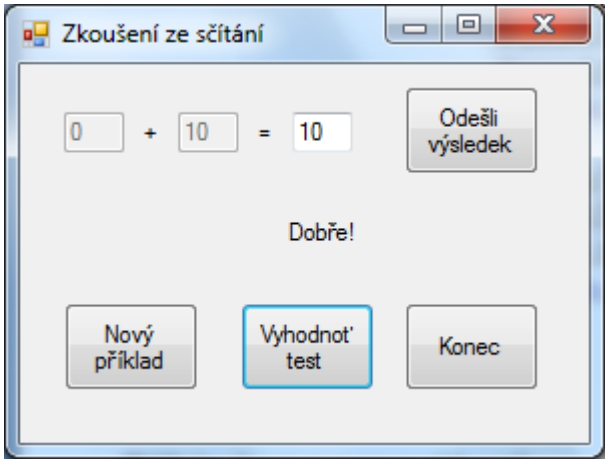
stav 0 stav 1 stav2 stav 3

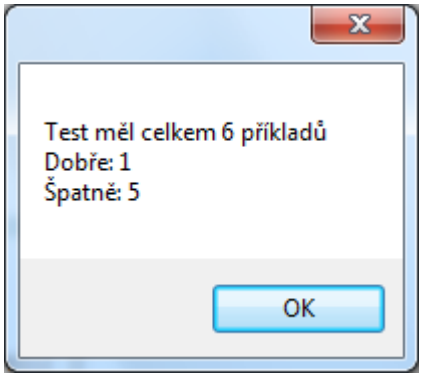
Použijeme členskou proměnnou typu int, kterou na začátku nastavíme na nulu. Po stisku tlačítka se tato porměnná zvětší o 1 a vypočítá se z ní zbytek po dělení 4. Podle hodnoty zbytku se pomocí konstrukce `switch` nastaví barva pozadí panelů.

```
Public partial class OknoProgramu : Form
{
   int počet = 0;
public OknoProgramu()
        {
            InitializeComponent();
        }
privatevoid tlačítkoPřepni_Click(object sender, EventArgs e)
    {
     počet++;
     int zbytek = počet % 4;
     switch (zbytek)
     {
          case 0:   panelČervená.BackColor = Color.Red;
                    PanelŽlutá.BackColor = Color.Gray;
                    PanelZelená.BackColor = Color.Gray;
          break;
          case 1:   panelČervená.BackColor = Color.Red;
                    PanelŽlutá.BackColor = Color.Yellow;
                    PanelZelená.BackColor = Color.Gray;
          break;
          case 2:   panelČervená.BackColor = Color.Gray;
                    PanelŽlutá.BackColor = Color.Gray;
                    PanelZelená.BackColor = Color.Green;
          break;
          case 3:   panelČervená.BackColor = Color.Gray;
                    PanelŽlutá.BackColor = Color.Yellow;
                    PanelZelená.BackColor = Color.Gray;
          break;
            }
      }
```

Příklady na procvičení

1. Napište program na sčítání od 0 do 20. (Generujte sčítance od 0 do 10).





2. Vylepšete program hrací kostka tak, aby zobrazoval „puntíky“ místo čísla.

11. Cyklus

Někdy potřebujeme provést stejnou činnost opakovaně. Abychom nemuseli psát mnohokrát za sebou stejné příkazy, použijeme **cyklus**. Často také dopředu nevíme, kolikrát se bude daná činnost opakovat a nemůžeme ji tedy zapsat jednotlivými příkazy.

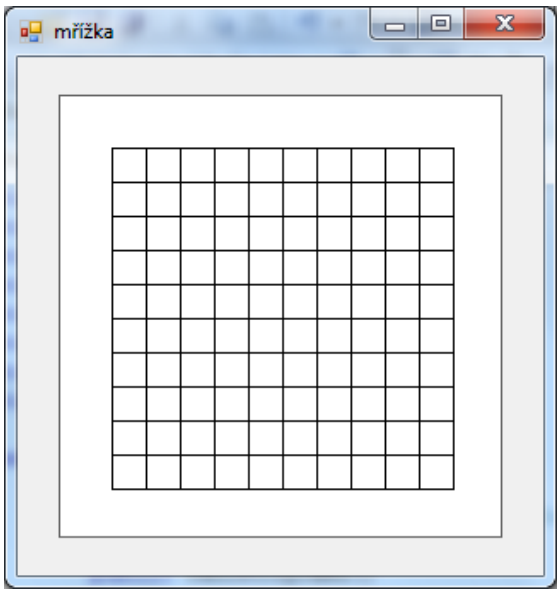
Cyklus s daným počtem opakování - řídicí konstrukce `for`

Tento cyklus je řízen proměnnou (obvykle celočíselnou), která se postupně mění (sčítáním nebo odčítáním) o stejnou hodnotu. Předem známe dolní a horní mez hodnoty řídicí proměnné, víme tedy kolikrát se bude cyklus opakovat.

```
for (inicializace; podmínka_opakování; aktualizace)
{
      příkazy;
}
```

Pokud je příkaz jen jeden, nepíšeme složené závorky.
Inicializace je deklarace a zadání počáteční hodnoty řídicí proměnné
Podmínka_opakování je obvykle ve tvaru porovnávání řídicí proměnné s konečnou hodnotou.
Aktualizace je změna hodnoty řídicí proměnné.

Př.
```
for (int x=1; x<=100; x+=1)
…
for (int x=10; x<=100; x+=10)
…
for (int x=100; x>=1; x-=1)
…
```
cyklus proběhne 100-krát
hodnota x se mění od 1 do 100 po jedné
cyklus proběhne 10-krát
hodnota x se mění od 10 do 100 po deseti
cyklus proběhne 100-krát
hodnota x se mění od 100 do 1 po jedné

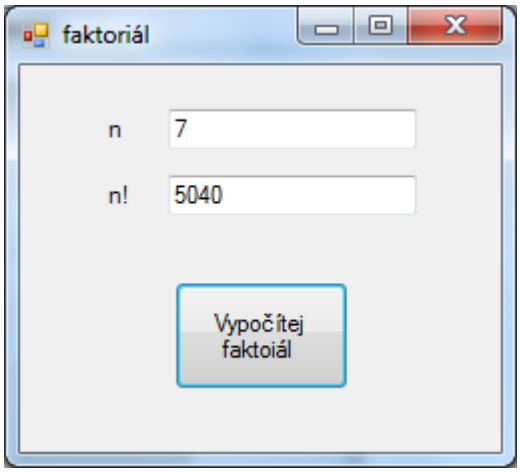


Př.

```
privatevoid panel1_Paint(object sender,
PaintEventArgs e)
{
Graphics kp = e.Graphics;
//svislé čáry
for (int x = 30; x <= 230; x += 20)
      kp.DrawLine(Pens.Black,x, 30, x, 230);
//vodorovné čáry
for (int y = 30; y <= 230; y += 20)
      kp.DrawLine(Pens.Black, 30, y, 230, y);
}
```

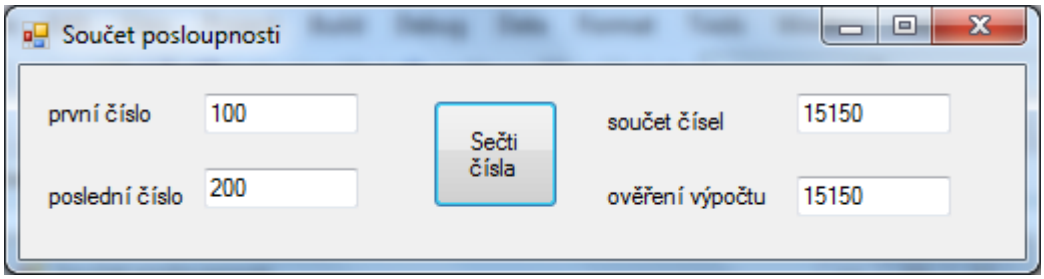
Kreslíme mřížku o deseti buňkách vodorovně a deseti svisle. Šířka a výška buňky je 20px.

Př. Výpočet faktoriálu.
Pro výsledek použijeme typ `double`, protože faktoriál velmi rychle roste a už pro číslo 13 dojde k „přetečení“ datového typu `int` a vracely by se nám chybné výsledky.



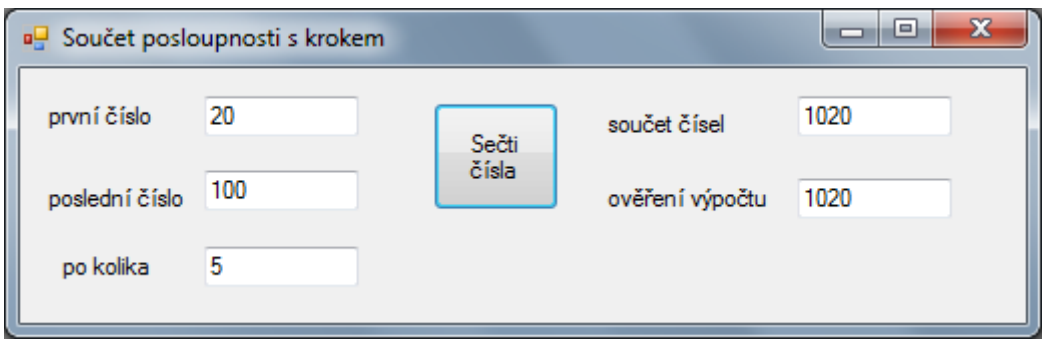

```
privatevoid TlačítkoVypočítej_Click(object
sender, EventArgs e)
 {
int n;
try
      { n = Convert.ToInt32(PoleN.Text); }
catch { return; }
double f = 1;
for (int x = 2; x <= n; x++)
      f *= x;
      PoleFaktoriál.Text = f.ToString("F0");
 }
```

Př. Součet čísel od... do …. Nejprve provedeme součet pomocí cyklu. Výpočet ověříme pomocí součtu členů aritmetické posloupnosti s diferencí 1.



Při výpočtu ověření musíme zvolit typ double, protože dělíme 2 a můžeme dostat desetinný výsledek (při lichém počtu čísel). Zároveň musíme dát pozor na celočíselné dělení a zapsat podíl `počet / 2.0`

```
privatevoid TlačítkoSečti_Click(object sender, EventArgs e)
{
      int první, poslední;
      try
         {
            první = Convert.ToInt32(PolePrvní.Text);
            poslední = Convert.ToInt32(PolePoslední.Text);
         }
      catch { return; }
      int součet = 0;
      for (int x = první; x <= poslední; x++)
        součet += x;
      int počet=poslední-první+1;
      double ověření = (počet / 2.0) * (první + poslední);
      PoleSoučet.Text = součet.ToString("F0");
      PoleOvěření.Text = ověření.ToString("F0");
}
```



V druhé verzi přidáme variantu „s krokem“. Čísla nepůjdou za sebou po jedné, ale o hodnotu, kterou zadáme, jako krok.
Vstupní data nejsou ošetřena ověřením, zda konečné číslo odpovídá danému kroku, pokud to tak není, tak se cyklem nezapočítá poslední číslo a ověřovací výsledek může být chybný.

```
privatevoid TlačítkoSečti_Click(object sender, EventArgs e)
{
int první, poslední, krok;
try
  {
      první = Convert.ToInt32(PolePrvní.Text);
      poslední = Convert.ToInt32(PolePoslední.Text);
      krok = Convert.ToInt32(PoleKrok.Text);
   }
catch { return; }
int součet = 0;
for (int x = první; x <= poslední; x+=krok)
      součet += x;
int počet=(poslední-první+krok)/krok;
double ověření = (počet / 2.0) * (první + poslední);
 PoleSoučet.Text = součet.ToString("F0");
   PoleOvěření.Text = ověření.ToString("F0");
}
```

Cyklus s podmínkou na začátku – řídicí konstrukce `while`

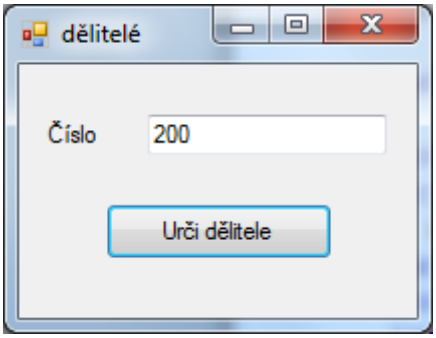
```
while (podmínka_opakování)
{
     příkazy;
}
```

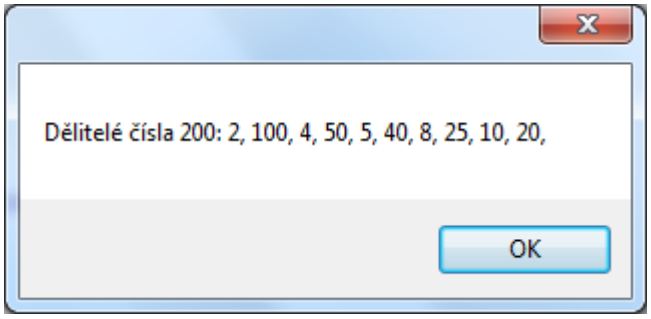
Příkazy uvedené v těle cyklu se provádí tak dlouho, dokud je splněna *podmínka_opakování*. V těle cyklu musí být příkaz, který dává šanci změnit výsledek podmínky, jinak bychom získali nekonečnou smyčku. Podmínka se testuje na začátku, není-li splněna už při prvním testování, cyklus vůbec neproběhne.

Př. Zápis metody `TlačítkoVypočítej_Click` z příkladu na výpočet faktoriálu. Tentokrát řešeno pomocí cyklu `while`. To, co řešíme `for` cyklem, vždy můžeme řešit i cyklem `while`.

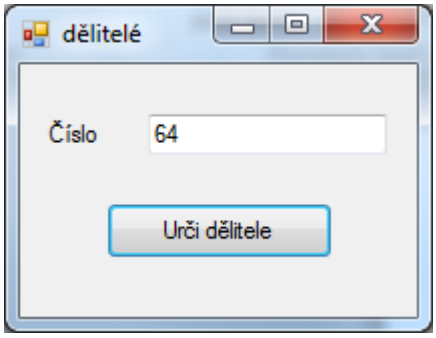
```
privatevoid TlačítkoVypočítej_Click(object sender, EventArgs e)
 {
int n;
try
   { n = Convert.ToInt32(PoleN.Text); }
catch { return; }
double f = 1;
int x=2;
while (x <= n)
   {
      f *= x;
      x++;
   }
   PoleFaktoriál.Text = f.ToString("F0");
 }
```

Př.

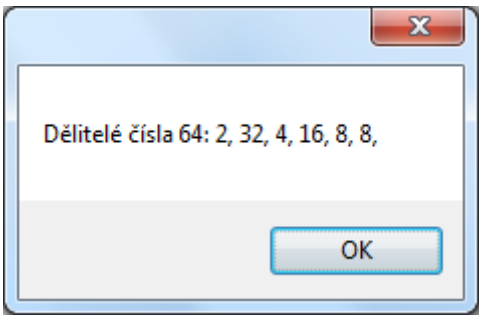





```
privatevoid TlačítkoUrčiDělitele_Click(object sender, EventArgs e)
{
int číslo;
string zpráva;
try
     { číslo = Convert.ToInt32(PoleČíslo.Text); }
catch {return;}
if (číslo>=0)
    {
        zpráva="Dělitelé čísla "+číslo.ToString()+": ";
int odmocnina=(int) Math.Sqrt(číslo);
int dělitel=2;
while (dělitel<=odmocnina)
        {
if (číslo % dělitel == 0)
           {
           zpráva += dělitel.ToString() + ", ";
               zpráva += (číslo / dělitel).ToString() + ", ";
           }
           dělitel++;
        }
      }
else zpráva="Pro záporné číslo nemá příklad smysl";
MessageBox.Show(zpráva);
}
```



Při zjišťování dělitelů daného čísla dělíme toto číslo postupně od dvou do odmocniny. Pokud je zbytek po dělení nula, našli jsme dva dělitele, číslo, kterým dělíme a podíl. Mírnou vadou tohoto způsobu řešení je, že pokud má číslo přirozenou odmocninu, pak se tato odmocnina napíše dvakrát. Pokusný blok `try-catch` zabrání běhové chybě, takže vždy bude zadáno celé číslo, tedy i záporné. Pro záporné číslo nemá příklad smysl řešit, proto je zde podmínka `if (číslo>=0)`… Podobně jsme měli ošetřit i příklad s faktoriálem.

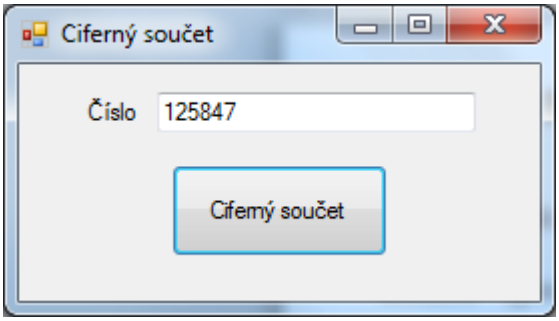


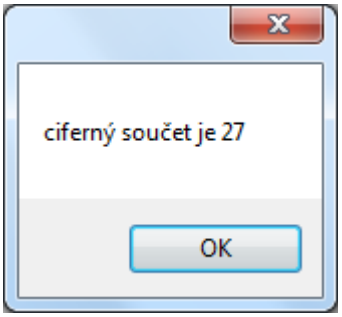
Cyklus s podmínkou na konci – řídicí konstrukce `do-while`

```
do
{
     příkazy;
}
while (podmínka_opakování);
```

Příkazy uvedené v těle cyklu se provádí tak dlouho, dokud je splněna *podmínka_opakování*.
Podmínka se testuje na konci, proto cyklus proběhne vždy alespoň jednou. Cyklem do-while tedy nelze řešit situace, kdy existuje možnost, že cyklus vůbec neproběhne. Např. u faktoriálu, pokud násobíme od 2, tak pro n=0 nebo n=1, vůbec nenásobíme.

Př.






```
privatevoid TlačítkoCifernýSoučet_Click(object sender, EventArgs e)
{
int číslo;
string zpráva;
try
    { číslo = Convert.ToInt32(PoleČíslo.Text); }
catch { return; }
if (číslo >= 0)
    {
int součet = 0;
do
        {
            součet+=číslo%10;
            číslo=číslo/10;
        }
while (číslo > 0);
        zpráva = "ciferný součet je " + součet.ToString();
    }
else zpráva = "Pro záporné číslo nemá příklad smysl";
MessageBox.Show(zpráva);
}
```

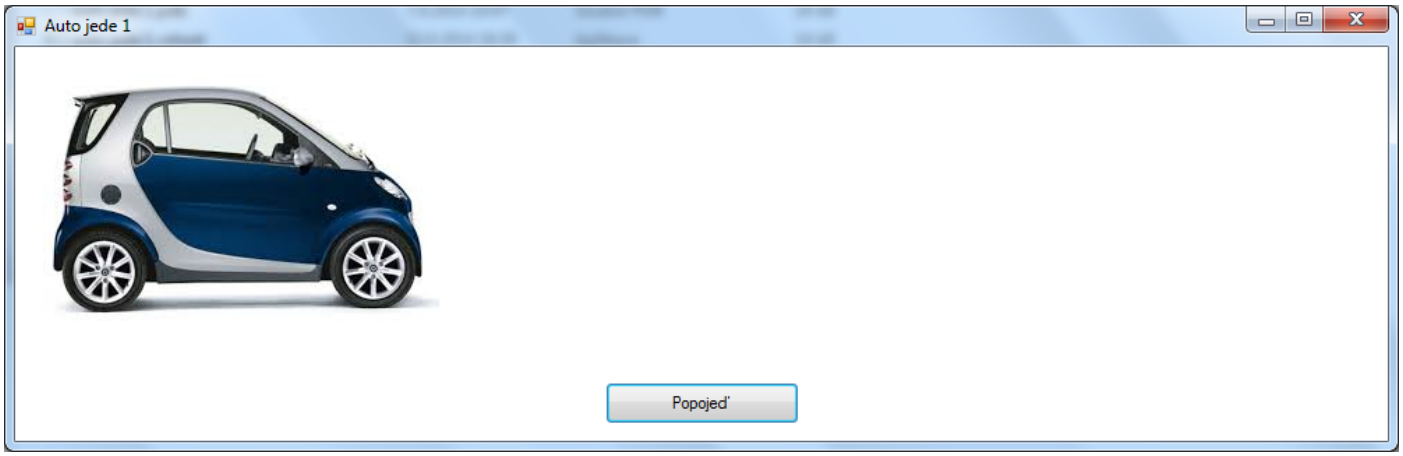
Tento příklad svádí k zadání velkého čísla. Pozor na přetečení! Typ int32 má rozsah $\pm 2^{31}$, tj. ±2147483648.

12. Animace a pohyb

„Ospalé“ auto

V následujících příkladech budeme pohybovat obrázkem autíčka. Nejprve použijeme prostředky, které již známe a využijeme cyklus s daným počtem opakování a „uspávání “ programu.

- Do okna programu budeme vykreslovat obrázek auta, který ve formě vestavěných dat zařadíme do projektu. Obrázek budeme kreslit v metodě `OknoProgramu_Paint`pomocí metody `DrawImage`. Souřadnice levého horního rohu obrázku budeme uchovávat v členských proměnných `x,y`.
- Při změně pozice auta vodorovně zleva doprava budeme zvětšovat proměnnou `x` o 2 pixely. Vykreslení auta v nové pozici vyvoláme voláním metody `Refresh`.
- Při stisku tlačítka „Popojeď“ se 50-krát zopakuje cyklus, ve kterém nejdříve posuneme a vykreslíme auto a pak na 50ms „uspíme“ program.




```
publicpartialclassOknoProgramu : Form
    {
          int x = 20;
          int y = 20;
          Image auto = Properties.Resources.auto;

public OknoProgramu()
   {
          InitializeComponent();
   }

privatevoid TlačítkoPopojeď_Click(object sender, EventArgs e)
   {
     for (int i = 0; i < 50; i++)
     {
        x += 2;
        Refresh();
        System.Threading.Thread.Sleep(50);
     }
   }
privatevoid OknoProgramu_Paint(object sender, PaintEventArgs e)
  {
     Graphics kp = e.Graphics;
     kp.DrawImage(auto, x, y);
  }
}
```

Po spuštění programu zjistíme, že autíčko sice plynule popojede, v našem případě o 100px, ale při vykreslování auta dochází k viditelnému problikávání obrazu.
Tento problém odstraníme technikou **zdvojení obrazové paměti**. Dosáhneme toho tím, že pro okno programu nastavíme vlastnost `DoubleBuffered` na `True`. Tutovlastnost lze nastavit pouze pro okno programu, nikoliv pro panel, proto budeme animace vytvářet v okně programu.

Časovač

Druhým nedostatkem posouvání auta „uspáváním“ je zablokování zbytku programu po dobu animace.
K odstranění je třeba změnit princip posunu obrázku a „uspávání se zcela vyhnout. Dosáhneme toho použitím nového ovládacího prvku – **časovače**.
Časovač je objekt, který umí v daném časovém intervalu generovat událost `Tick` – „tiká“. Do programu dostaneme časovač přetažením z Toolboxu (kategorie Components, prvek se jmenuje Timer) nad okno programu. Protože není vidět přímo v okně spuštěného programu, zobrazí se ve spodní části Designeru, pod oknem programu.
Vlastnosti časovače, které budeme nastavovat (kromě jména) budou:

- `Enabled` – určuje zda běží (`True`) nebo stojí (`False`).
- `Interval` – určuje dobu mezi dvěma „tiky“.

Použití časovače nám umožní auto rozjet a kdykoliv zastavit. Nemusíme s ním pouze popojíždět.

- Tlačítko START spustí časovač. Při každém „tiknutí“ se změní x-ová pozice a okno se překreslí. Časovač bude „tikat“ s intervalem 50ms, tedy 20-krát za sekundu.
- Tlačítko STOP vypne časovač

```
publicpartialclassOknoProgramu : Form
    {
int x = 20;
int y=20;
Image auto = Properties.Resources.auto;

public OknoProgramu()
        {
            InitializeComponent();
        }

privatevoid TlačítkoStart_Click(object sender, EventArgs e)
        {
            timer.Enabled = true;

        }

privatevoid TlačítkoStop_Click(object sender, EventArgs e)
        {
            timer.Enabled = false;
        }

privatevoid OknoProgramu_Paint(object sender, PaintEventArgs e)
        {
Graphics kp = e.Graphics;
            kp.DrawImage(auto, x, y);
        }

privatevoid timer_Tick(object sender, EventArgs e)
        {
            x += 2;
            Refresh();
        }
    }
```

Rychlost

V obou předchozích variantách příkladu, jsme auto posouvali o 2px každých 50ms. Jelo tedy rychlostí 40px/s. Změnit rychlost pohybu je možné dvěma způsoby:

- Budeme **měnit interval časovače**. - Pokud bychom měli více pohybujících se objektů, musel by mít každý svůj časovač, nebo by se měnila rychlost všem najednou. Interval časovače navíc z technických důvodů není vhodné nastavit na o mnoho nižší hodnotu než 50ms.
- Budeme **měnit počet pixelů**o který objekt posuneme. – Tato možnost neumožňuje plynulé změny rychlosti. Jen při přechodu z 2px na 3px se rychlost zvětší o 50%.

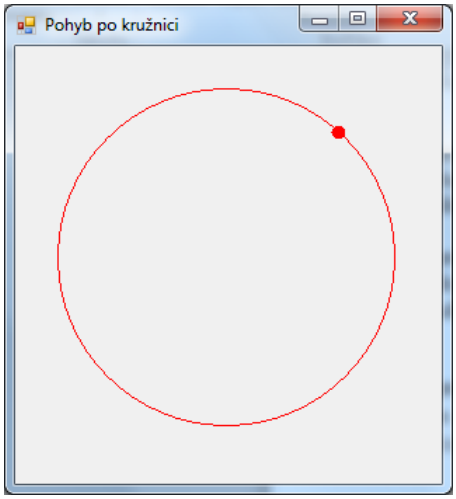
Abychom umožnili autu měnit rychlost, použijeme druhou variantu – změnu velikosti posunu. Změnu ovšem nebudeme provádět o celé pixely. Členskou proměnnou `x` zavedem jako reálnou, tedy typu double. Až při vykreslování auta do okna programu provedeme zaokrouhlení na celé pixely a vytvoříme celočíselnou proměnnou `xp`, která bude určovat pozici vykreslovaného objektu.
Velikost změny souřadnice x je vlastně **dráha**, kterou urazí objekt při jednom „tiku“. Určíme ji pomocí vztahu $s = v \cdot t$. Kde ***v*** je **rychlost** pohybu v px/s a ***t*** je **čas** jednoho „tiku“ v sekundách. Čas vypočítáme vynásobením intervalu časovače číslem 0,001.

```
publicpartialclassOknoProgramu : Form
    {
double x = 20;
int yp = 20;
double rychlost = 40;
public OknoProgramu()
        {
            InitializeComponent();
        }

privatevoid časovač_Tick(object sender, EventArgs e)
        {
double čas = 0.001 * časovač.Interval;
            x += rychlost * čas;
            Refresh();
        }

privatevoid TlačítkoStart_Click(object sender, EventArgs e)
        {
            časovač.Enabled = true;
        }

privatevoid TlačítkoStop_Click(object sender, EventArgs e)
        {
            časovač.Enabled = false;
        }

privatevoid OknoProgramu_Paint(object sender, PaintEventArgs e)
        {
Graphics kp = e.Graphics;
int xp = Convert.ToInt32(x);
            kp.DrawImage(Properties.Resources.auto, xp, yp);
        }
    }
```

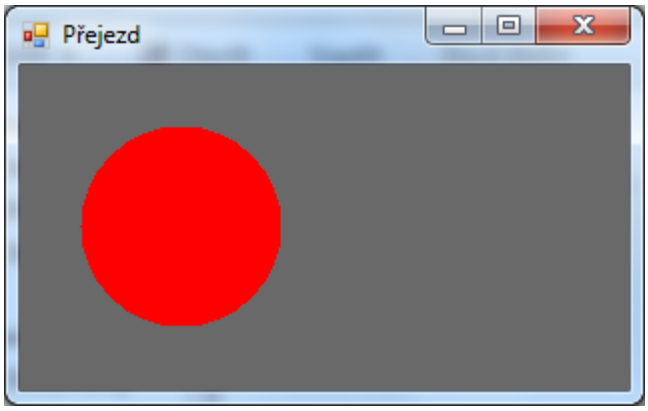
Příklady na procvičení

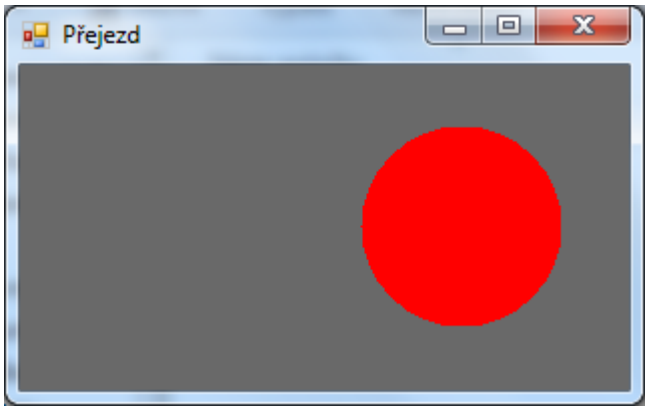
1. Napište program **fotogalerie**, v němž se bude v okně programu střídat několik (třeba 5) fotek po dvou sekundách.

2. Napište program **přímočarý pohyb**. Při stisku tlačítka se objeví uprostřed okna nová kulička, která se začne pohybovat ven z okna náhodně zvoleným směrem. Směr pohybu určíte jako vektor rychlosti – budete náhodně volit rychlost ve směru osy x a ve směru osy y. Obě hodnoty vybírejte z intervalu -100 až +100.

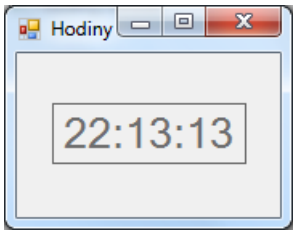


3. Napište program **přejezd**, který bude simulovat blikání železničního přejezdu. Po půlsekundách bude v okně přeskakovat červené kolečko. Můžete doplnit i zvukem.





4.Napište program **pohyb po kružnici**. V členských proměnných uchovávejte souřadnice středu kružnice a středu kuličky. Rychlost pohybu udáváω - úhlová rychlost v stupních za sekundu. Pro výpočet středu kuličky využijte funkce `Math.Cos` a `Math.Sin`. Nezapomeňte na převod úhlu ze stupňů na radiány.



5. Napište program **digitální hodiny**. Pro práci s časem užijte instanci `DateTime`. Aktuální datum a čas zjistíte z vlastnosti `DateTime.Now`. Užijte vlastnosti `Hour`, `Minute` a `Second`.

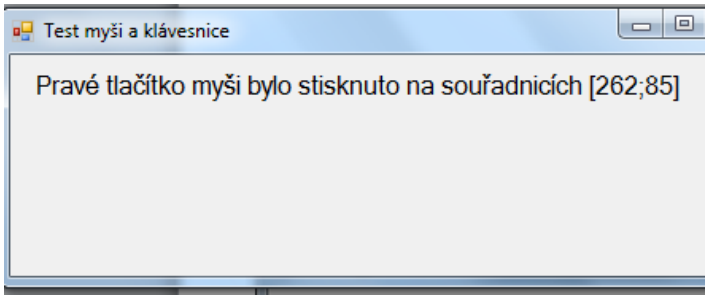
13. Myš a klávesnice

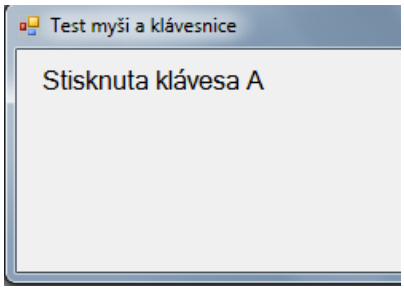
Test myši a klávesnice

Vytvoříme nový projekt, který nazveme *Test myši a klávesnice*. Do horního levého rohu okna umístíme popisek s názvem *Info* a prázdným textem.

Po stisku myši (nad oknem programu) se v popisku *Info* objeví hláška o tom, které tlačítko bylo stisknuto a na jaké pozici. Využijeme událost okna programu `MouseDown`. Souřadnice vyčteme z `e.X` a `e.Y`. Název tlačítka z `e.Button`.

Po stisku libovolné klávesy (je-li aktivní okno programu) se v popisku *Info* objeví hláška o tom, která klávesa byla stisknuta. Využijeme událost okna programu `KeyDown`. Název klávesy vyčteme z `e.KeyCode`.

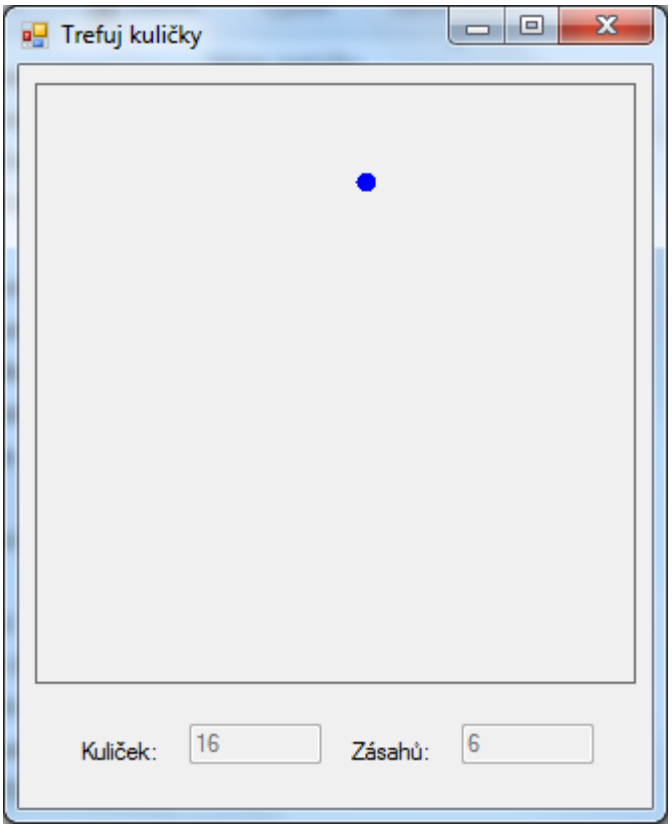




```
publicpartialclassOknoProgramu : Form
{
      public OknoProgramu()
          {
              InitializeComponent();
          }

      privatevoid OknoProgramu_MouseDown(object sender, MouseEventArgs e)
      {
            string tlačítko;
            switch (e.Button)
        {
            case MouseButtons.Left: tlačítko = "Levé";
            break;
            caseMouseButtons.Right: tlačítko = "Pravé";
            break;
            caseMouseButtons.Middle: tlačítko = "Prostřední";
            break;
            default: tlačítko="Neznámé";
            break;
               }
            Info.Text=tlačítko+" tlačítko myši bylo stisknuto na
            souřadnicích ["+e.X.ToString()+";"+e.Y.ToString()+"]";
        }
      privatevoid OknoProgramu_KeyDown(object sender, KeyEventArgs e)
        {
            string klávesa = e.KeyCode.ToString();
            Info.Text = "Stisknuta klávesa " + klávesa;
        }
    }
}
```

Konzumace kláves

Z předchozího příkladu se může zdát, že ovládání programu z klávesnice bude velmi jednoduché. Při používání tohoto způsobu ovládání programu záhy zjistíme, že problém nastane v okamžiku, kdy v okně programu bude nějaký ovládací prvek. Tetnto prvek totiž zareaguje na stisk klávesy sám a na okno se už nedostane. Odstranění tohoto „kradení kláves“ dosáhneme nastavením vlastnosti okna `KeyPreview` na `True`.



Program *Trefuj kuličky*

Vytvoříme jednoduchou hru. Do panelu se budou v určitém časovém intervalu vykreslovat kuličky na náhodnou pozici. Myší budeme kuličky trefovat. Po zásahu změní kulička barvu. Změna barvy bude indikátorem zásahu a zároveň prostředkem k zablokování kuličky, abychom nemohli jednu kuličku trefit víckrát.

Budeme potřebovat členské proměnné na: počet kuliček, počet zásahů, souřadnice středu kuličky, barvu kuličky a generátor náhodných čísel.
Událost panelu `Paint` – vykreslí se kulička na dané pozici a v dané barvě.

Událost časovače `Tick` – vygenerují se nové souřadnice kuličky a nastaví se barva modrá
– pře kreslí se panel a zvětší o 1 počítadlo kuliček

Událost panelu `MouseDown` – vyhodnotí se pozice myši a zda to bylo nad kuličkou (užijeme vztah pro výpočet vzdálenosti dvou bodů – myši a středu kuličky). Zásah se počítá pouze u modré kuličky.
- při zásahu se zvětší o 1 počítadlo zásahů a nastaví se barva červená

```
Public partial classOknoProgramu : Form
{
          int x = -20;
          int y = -20;
          Color barva = Color.Blue;
          Random náhoda = new Random();
          int početKuliček = 0;
          int početZásahů = 0;
          public OknoProgramu()
    {
             InitializeComponent();
    }
privatevoid panelKuliček_Paint(object sender, PaintEventArgs e)
    {
          Graphics kp = e.Graphics;
          Brush štětec=newSolidBrush(barva);
         kp.FillEllipse(štětec, x-5, y-5, 10, 10);
    }
privatevoid časovač_Tick(object sender, EventArgs e)
    {
         x = náhoda.Next(5, panelKuliček.Width-5);
         y = náhoda.Next(5, panelKuliček.Height - 5);
         barva = Color.Blue;
         panelKuliček.Refresh();
         početKuliček++;
         PoleKuliček.Text = početKuliček.ToString();
    }
privatevoid panelKuliček_MouseDown(object sender,MouseEventArgs e)
    {
          if (barva == Color.Blue)
         {
               int rozdílX = e.X - x;
               int rozdílY = e.Y - y;
               double vzdálenost =
               Math.Sqrt(rozdílX * rozdílX + rozdílY * rozdílY);
               if (vzdálenost <= 5)
                {
                   barva = Color.Red;
                   panelKuliček.Refresh();
                   početZásahů++;
                   PoleZásahů.Text = početZásahů.ToString();
                }
         }
    }
}
```

Parametry obslužných metod

Při volání jakékoliv metod píšeme za názvem metody kulaté závorky a v nich seznam parametrů nebo také nic, pokud žádné parametry nejsou. Parametry jsou upřesňující informace, které se metodě předávají z místa jejího volání. Např. při volání `MessageBox.Show` se uvádí textový řetězec se zprávou, při volání `kp.DrawEllipse` se předává pero a čtyři číselné údaje, atd.

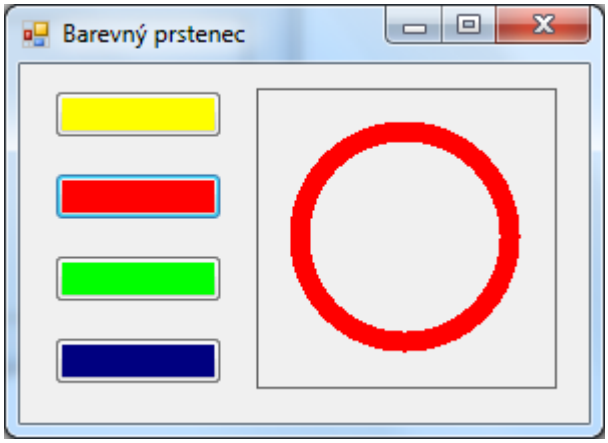
Při zápisu kódu metody se uvádí seznam parametrů, které metoda přebírá, v hlavičce metody.
Např. `private void panelKuliček_MouseDown(object sender,MouseEventArgs e)`
Tato metoda přebírá vždy dva parametry:

- parametr **sender** – odkaz na objekt, který událost vyvolal (v tomto případě panelKuliček).
- parametr **e** – upřesňující informace o dané události.

Při bližším zkoumání všech našich předchozích projektů zjistíme, že v hlavičkách námi napsaných metod jsou vždy tyto dva parametry. Parametr **sender** je vždy typu `object`, tj. jakýkoliv objekt. Typ parametru **e** závisí na konkrétí události. Ve výše uvedeném příkladu je typu `MouseEventArgs` a ukrává v sobě info o pozici myši a o tlačítku, které událost vyvolalo.
Parametr **e** jsme již dříve využívali při definici kreslícího plátna v události `Paint`.

Prametr sender

Parametr sender předává odkaz na objekt, který událost vyvolal. Této skutečnosti se využívá v případech, kdy metoda obsluhuje události více objektů (nejlépe stejného typu) najednou.



Př. Vytvoříme projekt *barevný prstenec*. Do panelu se bude vykreslovat barevná kružnice. Její barva se určí podle barvy pozadí tlačítka, které stiskneme. Abychom nemuseli pro každé tlačítko psát vlastní metodu `Click`, napíšeme jen jednu, kterou přejmenujeme (na názvu metody ve zdrojovém kódu stiskneme pravé tlačítko myši, zvolíme *Refactor – Rename*) a poté přiřadíme metodu všem ostatním tlačítkům, respektive jejich události `Click`.
Abychom identifikovali tlačítko, které bylo stisknuté a tím udává barvu, musíme vytvořit proměnnou typu `Button` a přiřadit jí parametr sender. Ten ovšem nejdřív musíme „přetypovat“ na typ `Button`.

```
Public partial classOknoProgramu : Form
 {
     Color barva = Color.Transparent;
     public OknoProgramu()
        {
            InitializeComponent();
        }

privatevoid panel_Paint(object sender, PaintEventArgs e)
    {
      Graphics kp = e.Graphics;
      Pen pero = new Pen(barva, 10);
      kp.DrawEllipse(pero,20,20,105,105);
     }

privatevoid buttons_Click(object sender, EventArgs e)
     {
        Button tlačítko = (Button)sender;
        barva = tlačítko.BackColor;
        panel.Refresh();
     }
}
```

Příklady na procvičení

1.Upravte dříve vytvořený projekt pohyblivý obrázek tak, aby se obrázek pohyboval po stisku šipek na klávesnici. (v archivu řešení – pohyblivý obrázek - klávesnice)

2. Upravte dříve vytvořený projekt auto jede tak, aby se auto rozjelo po stisku šipky doleva, zastavilo po stisku šipky doprava, zrychlilo šipkou nahoru a zpomalimo šipkou dolů. (v archivu řeší – auto jede 4).

3. Upravte připravený „polotovar“ tlačítka tak, aby všechna tlačítka při „najetí myší“ zčervenala a po „opuštění myší“ zase získala původní barvu (`Gold`). Užijte události `MouseEnter` a `MouseLeave`.

4. Upravte připravený „polotovar“ tlačítka tak, aby všechna tlačítka při kliknutí zmizela a po zmizení všech tlačítek se objevilo okno se zprávou a po jeho zavření program skončil. Tlačítkům nastavte hodnotu vlastnosti Visible na false. V členské proměnné si počítejte aktuální počet tlačítek, ať víte, kdy všechna zmizí.

5. Napište projekt *tažení myší“*. Do okna programu se vykreslí modrý čtverec, který půjde „táhnout“ po okně levým tlačítkem myši.

14. Pole

Pole je datová struktura s daným počtem (n) složek stejného typu, které se vzájemně rozlišují indexem.
V jazyce C# se indexuje celými čísly od 0.
Pole si můžeme představit jako n „chlívků“, kde v každém chlívku „bydlí“ hodnota daného typu a nad „chlívkem“ je označení číslem od 0 do n-1.

Příklad deklarace pole:

```
string [] jména = new string[5];
```

pole textových řetězců o 5 prvcích (0 až 4)

0	1	2	3	4
Tonda	Franta	Eva	Jára	Honza

```
int [] čísla = new int[20];
```

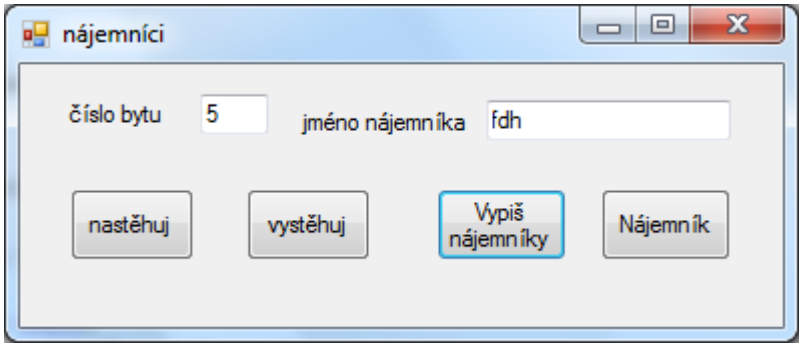
pole celých čísel o 20 prvcích (0 až 19)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
-5	12	0	14	125	-9	100	2	12	5	10	7	4	3	57	71	-35	13	1	9

Přístup k jednotlivým prvkům pole:
`jména[2]` je 3. prvek pole `jména` (prvek s indexem 2) tj. textový řetězec “Eva“.
`čísla[11]` je 12. prvek pole `čísla` (prvek s indexem 11) tj. číslo 7.

Program Nájemníci

Vytvoříme nový projekt. Budeme evidovat nájemníky v domě o deseti bytech.
K evidenci použijeme pole textových řetězců. Hodnotami v poli budou jména nájemníků.
Pro průchod celým polem (při výpisu) použijeme cyklus for.




```
Public partial class OknoProgramu : Form
{
     string[] nájemníci = new string[10];
public OknoProgramu()
        {
            InitializeComponent();
        }
```

```
privatevoid TlačítkoNastěhovat_Click(object sender, EventArgs e)
        {
          int čísloBytu;
          try { čísloBytu = Convert.ToInt32(PoleČíslo.Text); }
          catch { return; }
          if (čísloBytu > 0 && čísloBytu < 11)
            {
               nájemníci[čísloBytu - 1] = PoleJméno.Text;
               MessageBox.Show("Nájemník "+PoleJméno.Text+" nastěhován
                do bytu č. "+PoleČíslo.Text);
            }
          elseMessageBox.Show("chybně zadané číslo bytu");
        }
privatevoid TlačítkoVystěhuj_Click(object sender, EventArgs e)
      {
          int čísloBytu;
          try { čísloBytu = Convert.ToInt32(PoleČíslo.Text); }
          catch { return; }
          if (čísloBytu > 0 && čísloBytu < 11)
            {
                nájemníci[čísloBytu - 1] = null;
               MessageBox.Show("Byt č. " + PoleČíslo.Text+" je volný");
            }
          elseMessageBox.Show("chybně zadané číslo bytu");
      }
privatevoid TlačítkoVypiš_Click(object sender, EventArgs e)
        {
          string zpráva = "";
          for (int i = 1; i < 11; i++)
             zpráva += "byt " + i.ToString() + ": " + nájemníci[i-1]
                           + Environment.NewLine;
          MessageBox.Show(zpráva);
        }
privatevoid TlačítkoNájemník_Click(object sender, EventArgs e)
      {
          int čísloBytu;
          try { čísloBytu = Convert.ToInt32(PoleČíslo.Text); }
          catch { return; }
          if (čísloBytu > 0 && čísloBytu < 11)
            {
                PoleJméno.Text = nájemníci[čísloBytu - 1];
            }
          elseMessageBox.Show("chybně zadané číslo bytu");
        }
    }
```

V archivu příkladů je projekt označen jako *nájemníci1*.

Cyklus `foreach`

Při procházení celým polem je vhodnější místo cyklu **`for`**použít cyklus **`foreach`.**

```
privatevoid TlačítkoVypiš_Click(object sender, EventArgs e)
      {
          string zpráva = "";
          foreach (string nájemník innájemníci)
                zpráva += nájemník + Environment.NewLine;
```

```
            MessageBox.Show(zpráva);
        }
```

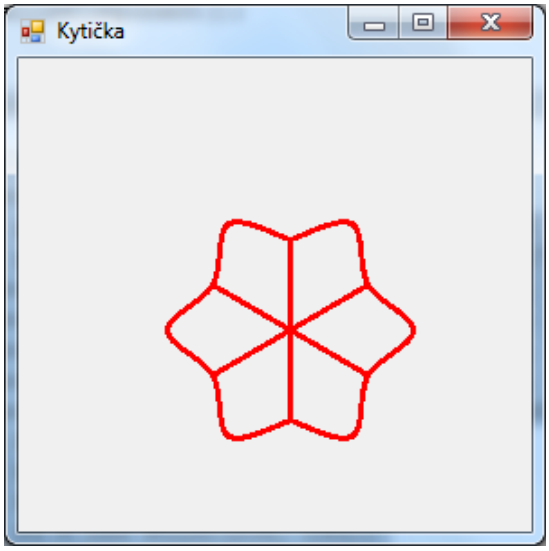
Toto je přepis metody pro výpis pole z příkladu nájemníci pomocí `foreach`. Musíme se tu obejít bez čísla bytu, takže v tomto případě použijeme raději trochu komplikovanější `for`, ale pro pochopení zápisu cyklu `foreach` to stačí.

Pole jako výstup

Datová struktura pole může být i výstupem některých metod, např. metody `GetLogicalDrives`, která vrací pole textových řetězců. Výstupní pole obsahuje názvy připojených diskových jednotek.

Př. Disky

```
privatevoid TlačítkoZjisti_Click(object sender,EventArgs e)
    {
      string[] disky = Environment.GetLogicalDrives();
      string zpráva = "";
      foreach (string disk in disky)
          zpráva += disk + Environment.NewLine;
       PoleDisky.Text = zpráva;
    }
```



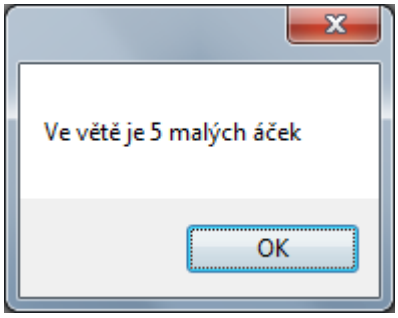
Pole jako vstup

Datový typ pole může být i vstupním parametrem některých metod, např. metody `DrawCurve`, která vykresluje křivku zadanou polem bodů.

Př. Kytička

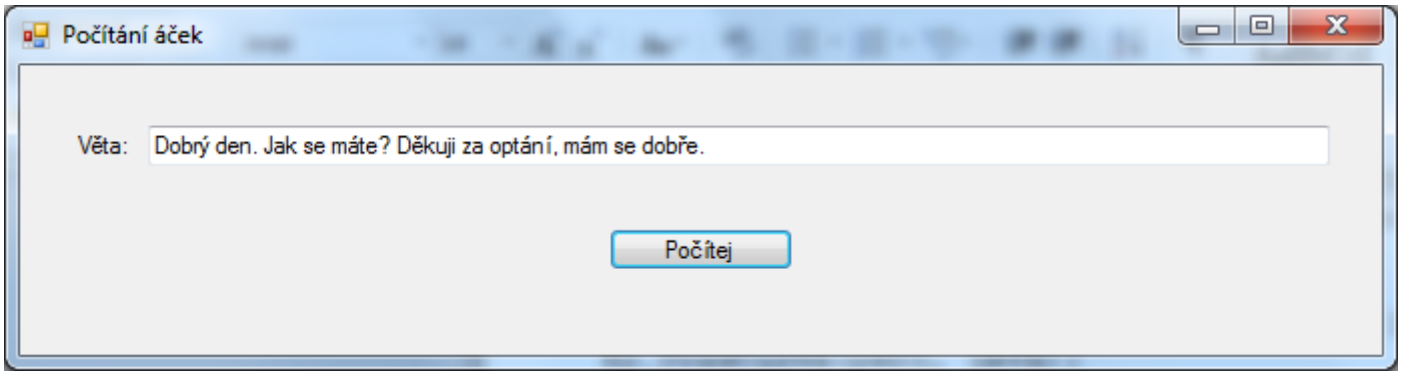
```
privatevoid OknoProgramu_Paint(object sender, PaintEventArgs e)
{
      Graphics kp = e.Graphics;
      Pen pero = new Pen(Color.Red, 3);
      int[] poleX = new int[] { 107, 150, 193 };
      int[] poleY = new int[] { 100, 125, 150, 175, 200 };
        kp.DrawLine(pero, poleX[0], poleY[1], 150, 150);
        kp.DrawLine(pero, poleX[0], poleY[3], 150, 150);
        kp.DrawLine(pero, poleX[2], poleY[1], 150, 150);
        kp.DrawLine(pero, poleX[2], poleY[3], 150, 150);
        kp.DrawLine(pero, poleX[1], poleY[0], poleX[1], poleY[4]);
      Point P1 = new Point(150, 100);
      Point P2 = new Point(184, 91);
      Point P3 = new Point(193, 125);
      Point P4 = new Point(218, 150);
      Point P5 = new Point(193, 175);
      Point P6 = new Point(184, 209);
      Point P7 = new Point(150, 200);
      Point[] pravá = new Point[] { P1, P2, P3, P4, P5, P6, P7 };
      kp.DrawCurve(pero, pravá);
      Point L1 = P1;
```

```
        Point L2 = new Point(116, 91);
        Point L3 = new Point(107, 125);
        Point L4 = new Point(82, 150);
        Point L5 = new Point(107, 175);
        Point L6 = new Point(116, 209);
        Point L7 = P7;
        Point[] levá = new Point[] { L1, L2, L3, L4,
L5, L6, L7 };
        kp.DrawCurve(pero, levá);
}
```



Řetězce jako pole znaků

Na textové řetězce můžeme pohlížet jako na pole znaků. Můžeme je procházet znak po znaku cyklem `for` nebo `foreach`. Toho využíváme tahdy, když chceme z pole pouze číst. Abychol mohli znaky i měnit, museli bycho použít buď instanci třídy `StringBuilder` nebo pole `char[]`.
Př. Počítání áček – dvě varianty 1 – s cyklem `for`, 2- s cyklem `foreach`.



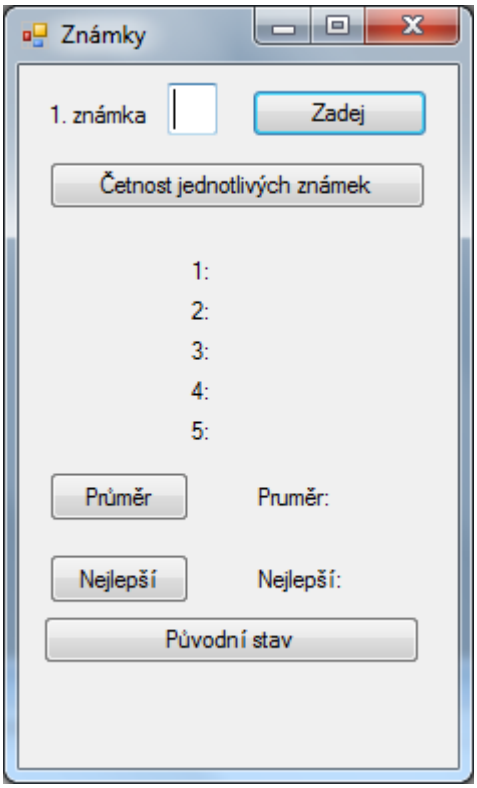

```
privatevoid TlačítkoPočítej_Click(object sender, EventArgs e)
{
        string Věta = PoleVěta.Text;
        int PočetZnaků = Věta.Length;
        int PočetÁček = 0;
        for (int čísloZnaku = 0; čísloZnaku < PočetZnaků; čísloZnaku++)
        if (Věta[čísloZnaku] == 'a' || Věta[čísloZnaku] == 'á')
            PočetÁček++;
 MessageBox.Show("Ve větě je " + PočetÁček.ToString() + " malých áček");
}

privatevoid TlačítkoPočítej_Click(object sender, EventArgs e)
{
        string Věta = PoleVěta.Text;
        int PočetÁček = 0;
        foreach (char písmeno in Věta)
        if (písmeno == 'a' || písmeno == 'á') PočetÁček++;
 MessageBox.Show("Ve větě je " + PočetÁček.ToString() + " malých áček");
}
```

Příklady na procvičení

1. Upravte projekt nájemníci. (v archivu příkladů – nájemníci2)

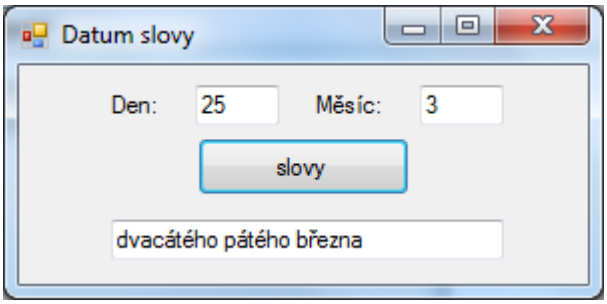
- přidejte možnost výpisu čísla bytu podle jména
- změňte nastěhování tak, aby musel byt být prázdný a není-li prázdný, zobrazí se zpráva
- změňte vystěhování tak, aby se zobrazila zpráva se jménem vystěhovaného

- přidejte tlačítko *kulový blesk*, které posune nájemníky v bytech o jedno číslo - z 1 do 2, z 2 do 3, …, z 10 do 1. Promyslete, jak by to bylo obráceně – z 2 do 1, z 3 do 2, …, z 1 do 10.
- přidejte tlačítko pro výpis prázdných bytů
- přidejte tlačítko pro výpis obsazených bytů



2. Napište program *Známky* pro výpočet statistických funkcí pro 20 známek.

3. Napište program *Datum slovy*, který přečte číselně zadané datum a převede ho na textový řetězec.



15. Soubory

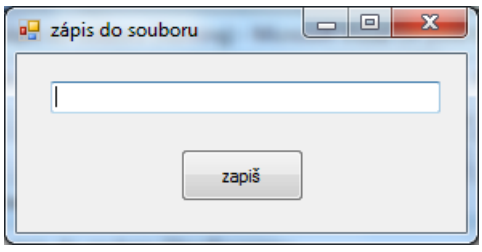
V této kapitole se budeme zabývat prací s textovými soubory, v kterých nejsou informace strukturované. Budou nás tedy zajímat soubory neformátovaného volného textu, takové které lze zpracovat např. v poznámkovém bloku. Jediné formátování textu v těchto souborech je oddělování řádků.
Pro práci se soubory musíme do horní části zdrojového kódu přidat řádek

```
using System.IO;
```

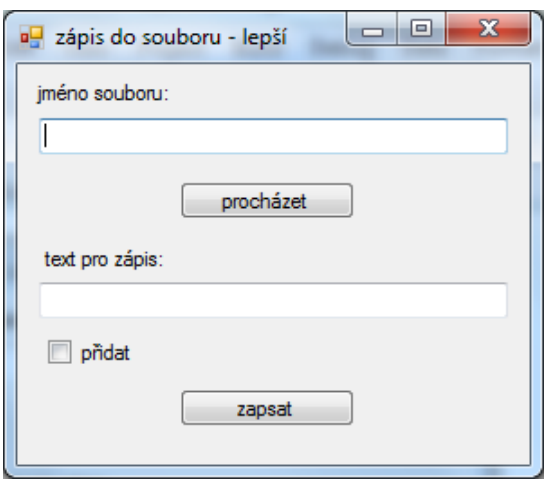
tím odkazujeme na jmenný prostor s třídami pro práci se soubory.
Práce se souborem probíhá ve třech částech:

1. otevření souboru – pro čtení nebo pro zápis
2. práce a daty
3. zavření souboru

Zápis do souboru



V tomto projektu budeme zapisovat text z textového pole do souboru `pokus.txt`. Soubor se vytvoří ve složce projektu v podsložce bin/debug. Pokud tento soubor již existuje, tak se přepíše.
Do souboru se zapisuje nadvakrát. Nejprve se zapíše text „Zápis textu: “ a pak se teprve zapíše a odřádkuje text z textového pole.

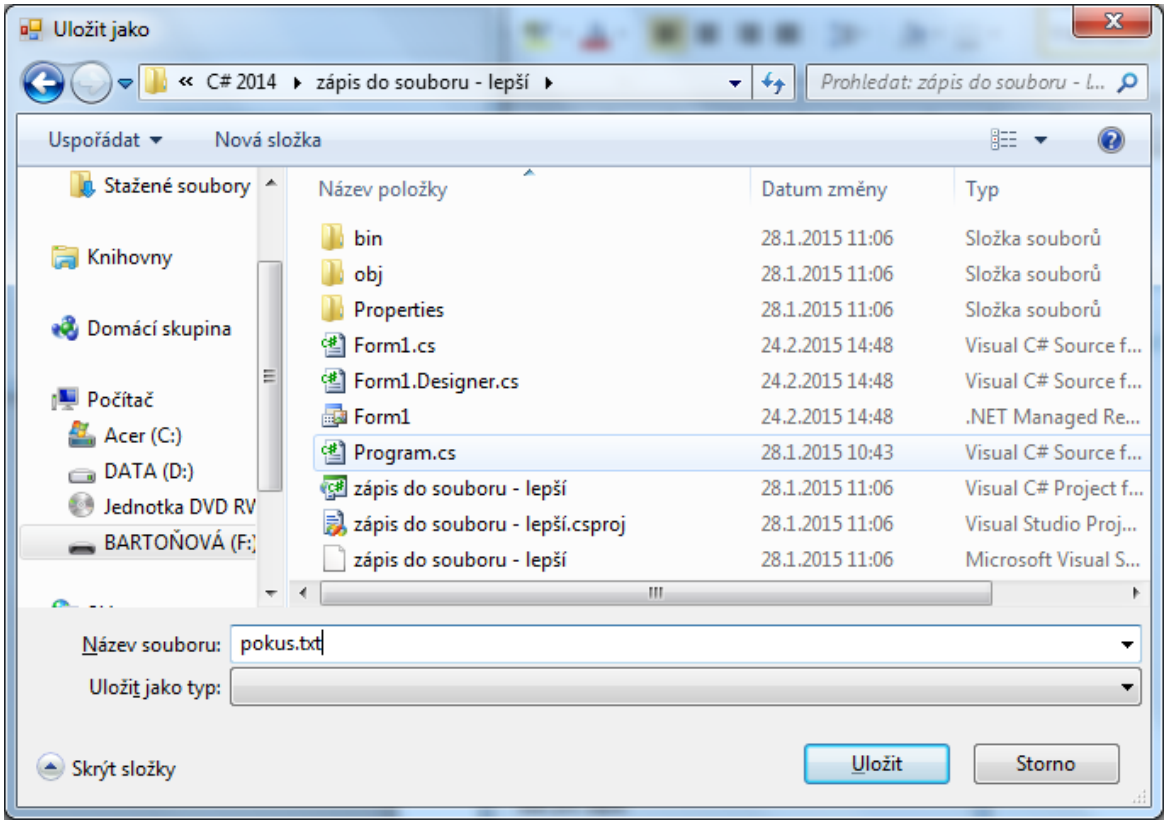



```
privatevoid button_Click(object sender,
EventArgs e)
{
StreamWriter soubor =
new StreamWriter("pokus.txt");
string řádek = textBox.Text;
```

```
    soubor.Write("Zápis textu: ");
    soubor.WriteLine(řádek);
    soubor.Close();
MessageBox.Show("hotovo");
}
```

V následujícím projektu došlo k několika vylepšením.

1. Pro výběr souboru je zde tlačítko procházet a používá se zde ovládací prvek `SaveFileDialog`, který najdeme v Toolboxu v kategorii Dialogs. Z Toolboxu jej přetáhneme nad okno designera, pustíme a objeví se nám pod oknem programu. Je to stejné jako použití časovače. Tento prvek je přejmenovaný na `oknoUloženíSouboru`. Po stisku tlačítka se pro `oknoUloženíSouboru`vyvolá metoda `ShowDialog()`. Tím vznikne dialogové okno pro výběr souboru pro zápis. Můžeme tím vytvořit nový soubor nebo zvolit již existující (pak jsme upozorněni, že soubor bude přepsán). Dojde-li k vybrání souboru a potvrzení tlačítkem uložit, tak metoda `ShowDialog()` vrátí hodnotu `OK` a díky následující podmínce je do textového pole`PoleJménoSouboru`zapsána cesta k vybranému souboru.




```
privatevoid TlačítkoProcházet_Click(object sender, EventArgs e)
      {
            DialogResult odpověď = oknoUloženíSouboru.ShowDialog();
            if (odpověď == DialogResult.OK)
                  PoleJménoSouboru.Text = oknoUloženíSouboru.FileName;
      }
```

2. Podle zaškrtnutí políčka „přidat“ vznikne logická proměnná `přidat`, kterou použijeme jako druhý argument při otevírání souboru. Je-li hodnota `true`, tak se text připíše na konec souboru a původní obsah bude zachován. Při hodnotě `false` dojde k přepsání textu (tak jako v předchozím příkladu).

3. Třetím argumentem při otevírání souboru je typ kódování. Hodnota `Encoding.Default`představuje standardní kódovou stránku Windows. Pro kódování UTF-8 bychom použili hodnotu `Encoding.UTF8`.

```
privatevoid TlačítkoZapsat_Click(object sender, EventArgs e)
{
string JménoSouboru = PoleJménoSouboru.Text;
bool přidat = checkBoxPřidat.Checked;
```

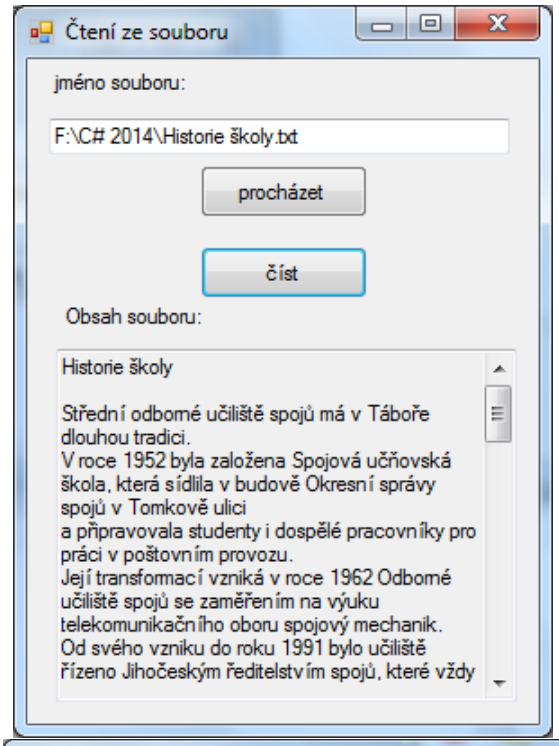
```
StreamWriter soubor = newStreamWriter(JménoSouboru, přidat,
    Encoding.Default);
string řádek = PoleZápis.Text;
    soubor.WriteLine(řádek);
    soubor.Close();
MessageBox.Show("hotovo");
}
```

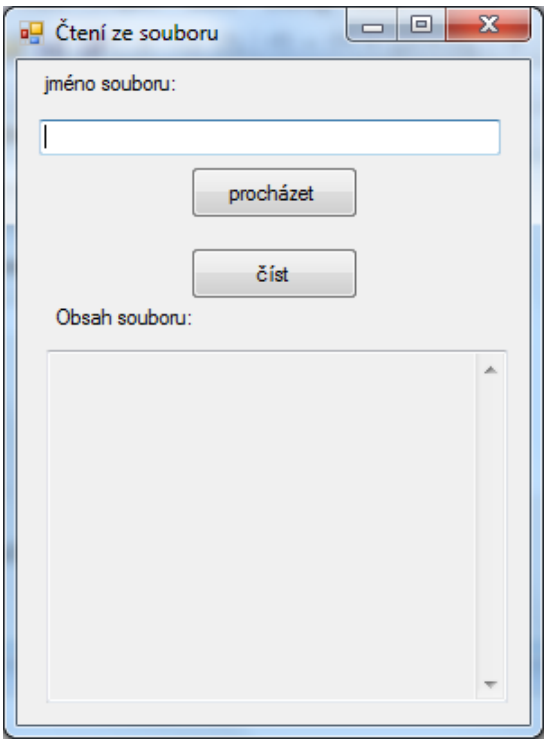
Čtení ze souboru

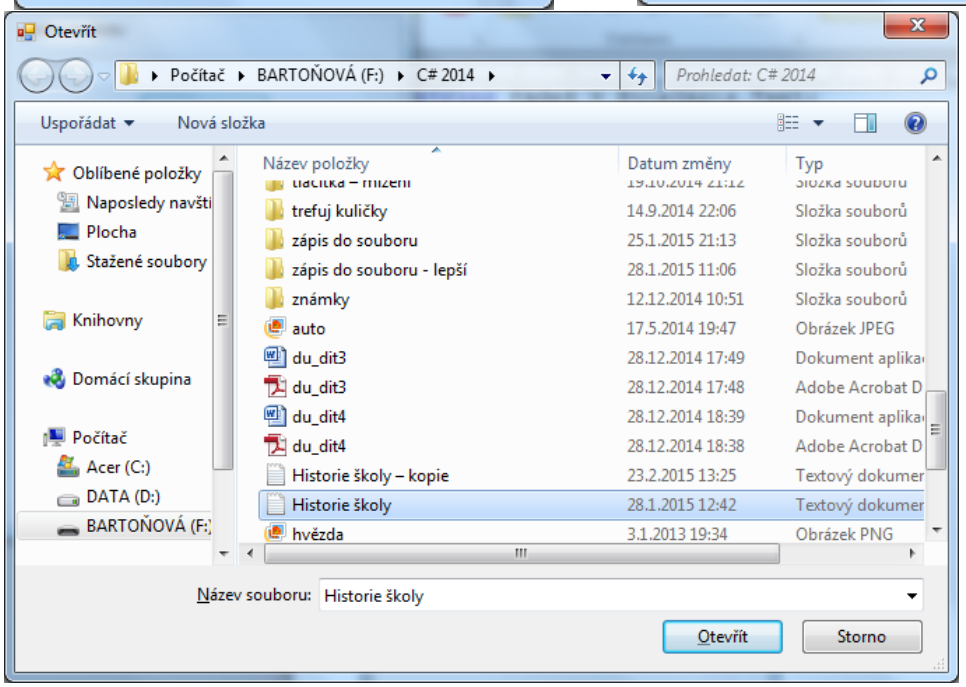
Čtení ze souboru probíhá podobně jako zápis.

Pro výběr souboru použijeme ovládací prvek `OpenFileDialog`, který je přejmenovaný na `oknoOtevřeníníSouboru`.

```
privatevoid tlačítkoProcházet_Click(object sender, EventArgs e)
  {
     DialogResult odpověď = OknoOtevřeníSouboru.ShowDialog();
     if (odpověď == DialogResult.OK)
              PoleJménoSouboru.Text = OknoOtevřeníSouboru.FileName;
  }
```







Pro otevření používáme `StreamReader(JménoSouboru,Encoding.Default);` a pro čtení celého řádku metodu `ReadLine()`.

Aby se četl celý soubor (řádek po řádku) musíme použít cyklus. Kdybychom věděli předem počet řádků, tak bychom použili cyklus `for`. My to ale většinou nevíme a musíme tak použít cyklus s podmínkou. Konec souboru se pozná podle toho, že metoda vrátí hodnotu null.

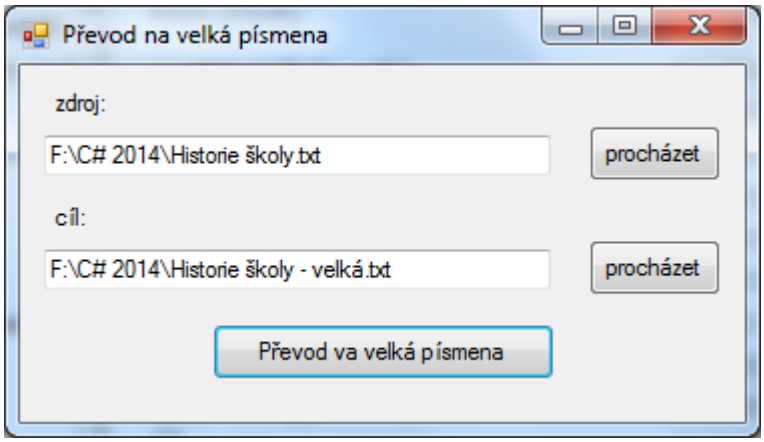
```
privatevoid TlačítkoČíst_Click(object sender, EventArgs e)
{
    string JménoSouboru = PoleJménoSouboru.Text;
    StreamReader soubor =
    new StreamReader(JménoSouboru,Encoding.Default);
    string řádek;
    PoleObsah.Text = null;
    do
    {
        řádek = soubor.ReadLine();
        PoleObsah.Text += řádek + Environment.NewLine;
    }
  while (řádek != null);
  soubor.Close();
}
```

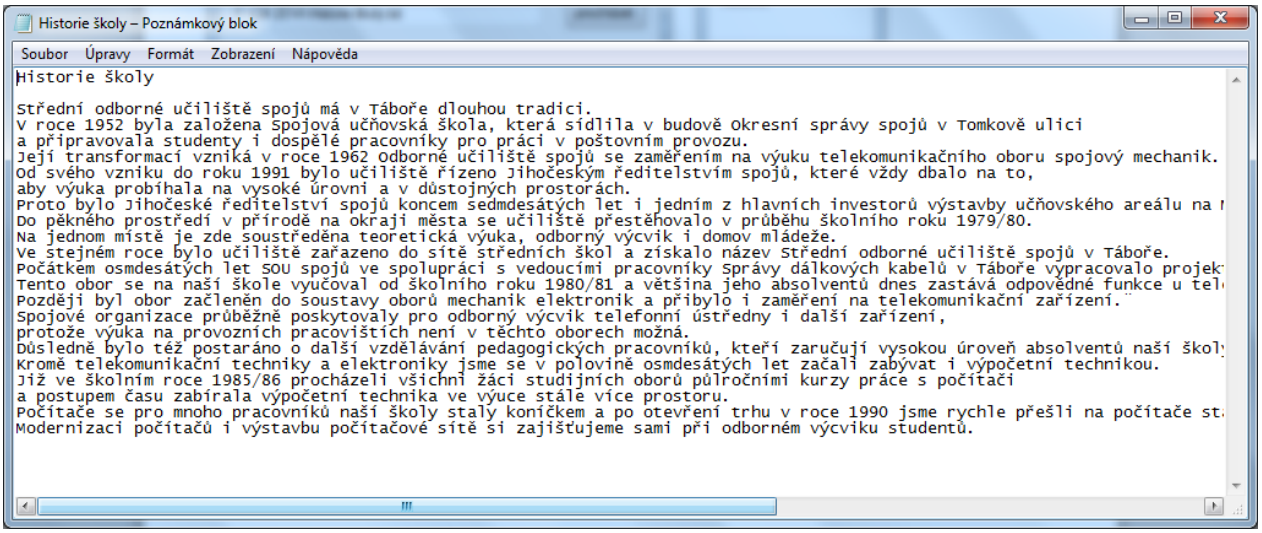
V tomto příkladu je použit cyklus do – while. Jako poslední se tedy zapisuje prázdný řádek. Tady nám to nevadí, ale pokud bychom s načteným textem dále pracovali, tak by to vadit mohlo a museli bychom mít podmínku na testování prázdného řádku. Vhodnější proto je cyklus while.

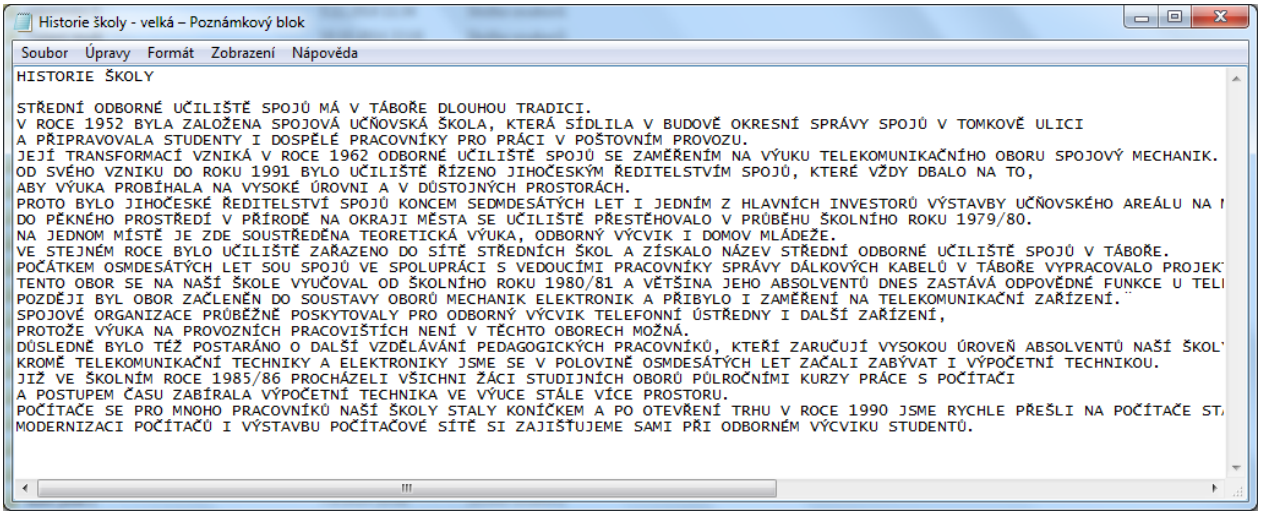
```
While((řádek = soubor.ReadLine()) != null)
    PoleObsah.Text += řádek + Environment.NewLine;
```

Převod na velká písmena

Po výběru obou souborů se přečte (řádek po řádku) obsah zdrojového souboru, každý řádek se převede na velká písmena a zapíše o cílového souboru.

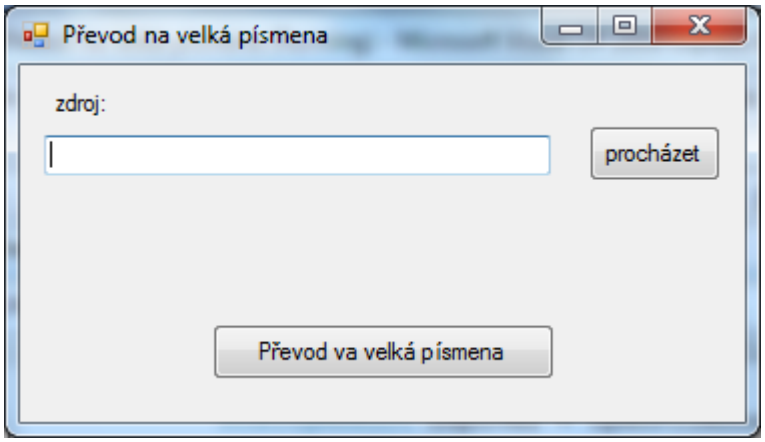






Jeden soubor

Pokud bychom chtěli provést převod na velká písmena v tom samém souboru, tak potřebujeme pomocný dočasný soubor. Náš soubor bude fungovat jako zdroj a dočasný jako cíl. Po ukončení převodu a uzavření obou souborů náš soubor smažeme a vytvoříme z dočasného nový se stejným jménem.




```
privatevoid tlačítkoProcházetZdroj_Click(object sender, EventArgs e)
{
     DialogResult odpověď = openFileDialog1.ShowDialog();
     if (odpověď == DialogResult.OK)
       PoleZdroj.Text = openFileDialog1.FileName;
}

privatevoid TlačítkoPřevod_Click(object sender, EventArgs e)
{
     string jménoSouboru = PoleZdroj.Text;
     string dočasnýAdresář = Path.GetTempPath();
     string náhodnéJméno = Path.GetRandomFileName();
     string dočasnýCíl = Path.Combine(dočasnýAdresář, náhodnéJméno);
     StreamReader zdroj = new StreamReader(jménoSouboru,
     Encoding.Default);
     StreamWriter cíl = new StreamWriter(dočasnýCíl, false,
     Encoding.Default);
     string řádek, novýřádek;
     while ((řádek=zdroj.ReadLine()) != null)
     {
         novýřádek = řádek.ToUpper();
         cíl.WriteLine(novýřádek);
      }
      zdroj.Close();
      cíl.Close();
     File.Delete(jménoSouboru);
     File.Move(dočasnýCíl,jménoSouboru);
     MessageBox.Show("hotovo");
}
```

16. Indexované seznamy

Přímý přístup k datům

Při zpracování dat z textového souboru může být problémem sekvenční přístup k datům. Abychom mohli přistupovat k danému záznamu, např. textovému řetězci, přímo, musíme si data ze souboru uložit do operační paměti.
Pokud bychom pro uložení dat chtěli použít pole, tak budeme muset nejprve zjistit, kolik prvků má dané pole mít. Musíme proto soubor číst dvakrát. Nejprve počítáme prvky, pak vytvoříme pole a poté při druhém čtení souboru načítáme prvky do pole.

Př. Ve složce projektu *bin/debug* máme soubor **jmena.txt**. Po stisku tlačítka se vytvoří soubor **obracene.txt**, do kterého se napíšou jména opačně než v původním souboru.

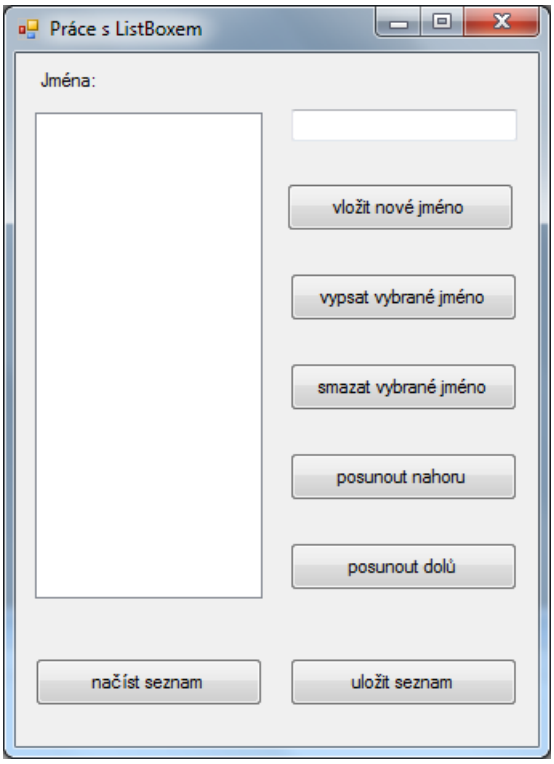
```
private void TlačítkoProveď_Click(object sender, EventArgs e)
{
   StreamReader zdroj = new StreamReader("jmena.txt", Encoding.Default);
   int početŘádků = 0;
   while (zdroj.ReadLine() != null)
      početŘádků++;
   zdroj.Close();
   string[] řádky = new string[početŘádků];
   zdroj = new StreamReader("jmena.txt", Encoding.Default);
   for (int i = 0; i < početŘádků; i++)
       řádky[i] = zdroj.ReadLine();
   zdroj.Close();
  StreamWriter cíl = new StreamWriter("obracene.txt", false,
  Encoding.Default);
  for (int i = početŘádků - 1; i >= 0; i--)
    cíl.WriteLine(řádky[i]);
  cíl.Close();
  MessageBox.Show("hotovo");
}
```

Vhodnější by bylo použít datovou strukturu, která může uchovat větší množství dat s přístupem pomocí indexu, ale abychom nemuseli dopředu znát počet prvků. Takovou strukturou je indexovaný seznam. Indexované seznamy jsou instancemi třídy `List<T>`, kde T je typ prvků seznamu. Při vytvoření seznamu konstruktorem `new` bez parametrů je seznam prázdný. Prvky se přidávají metodou `Add`. Aktuální počet prvků určuje vlastnost `Count`. U pole se tato vlastnost jmenuje `Length`. Pro práci se seznamy je potřeba jmenný prostor `System.Collections.Generic`. Pokud ve zdrojovém kódu není příslušný `using`, je ho třeba připojit.
Řešení předchozího příkladu pomocí seznamu `List<string>`.

```
private void TlačítkoProveď_Click(object sender, EventArgs e)
{
   List<string> řádky=new List<string>();
   StreamReader zdroj = new StreamReader("jmena.txt", Encoding.Default);
   string řádek;
   while ((řádek = zdroj.ReadLine()) != null)
      řádky.Add(řádek);
   zdroj.Close();
   StreamWriter cíl = new StreamWriter("obracene.txt", false,
   Encoding.Default);
   for (int i = řádky.Count-1; i >= 0; i--)
      cíl.WriteLine(řádky[i]);
   cíl.Close();
   MessageBox.Show("hotovo");
}
```

Operace nad seznamem

V následujícím příkladu použijeme místo seznamu `List<T>` ovládací prvek `ListBox`. Tento prvek je spojen se seznamem, který je přístupný přes vlastnost `Items`. V `ListBoxu` jsou prvky vidět.



Př.

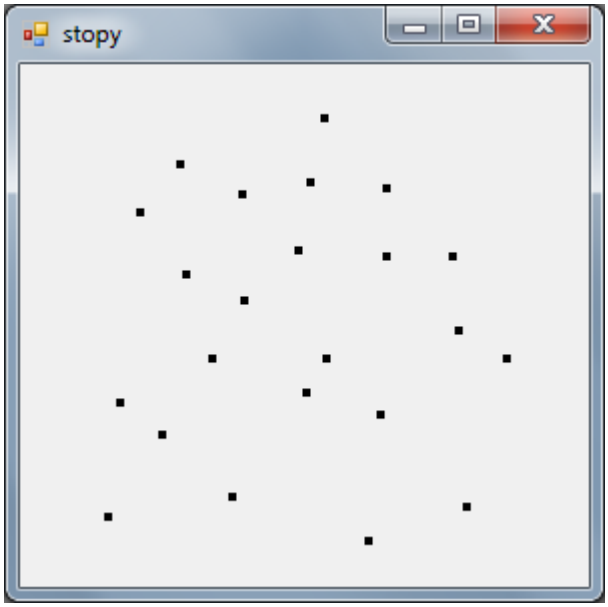
```
private void TlačítkoNačíst_Click(object
sender, EventArgs e)
{
   SeznamJmen.Items.Clear();
   try
   {
    StreamReader zdroj= new
    StreamReader("jmena.txt",
    Encoding.Default);
    string řádek;
    while ((řádek=zdroj.ReadLine())!=null)
          SeznamJmen.Items.Add(řádek);
   zdroj.Close();
}
 catch
{ MessageBox.Show("Chyba při načítání seznamu"); }
}

private void tlačítkoVypišVybrané_Click(object sender, EventArgs e)
{
    int index = SeznamJmen.SelectedIndex;
    if (index >= 0)
    MessageBox.Show(SeznamJmen.Items[index].ToString());
    else MessageBox.Show("Není vybrané žádné jméno");
}

private void TlačítkoVložNové_Click(object sender, EventArgs e)
{
    int index = SeznamJmen.SelectedIndex;
    if (index >= 0)
       SeznamJmen.Items.Insert(index, PoleNovéJméno.Text);
    else
       SeznamJmen.Items.Add(PoleNovéJméno.Text);
       PoleNovéJméno.Text = "";
}

private void tlačítkoUložit_Click(object sender, EventArgs e)
{
  StreamWriter cíl = new StreamWriter("jmena.txt", false,
  Encoding.Default);
  for (int index = 0; index < SeznamJmen.Items.Count; index++)
    cíl.WriteLine(SeznamJmen.Items[index]);
  cíl.Close();
  MessageBox.Show("hotovo!");
}
```

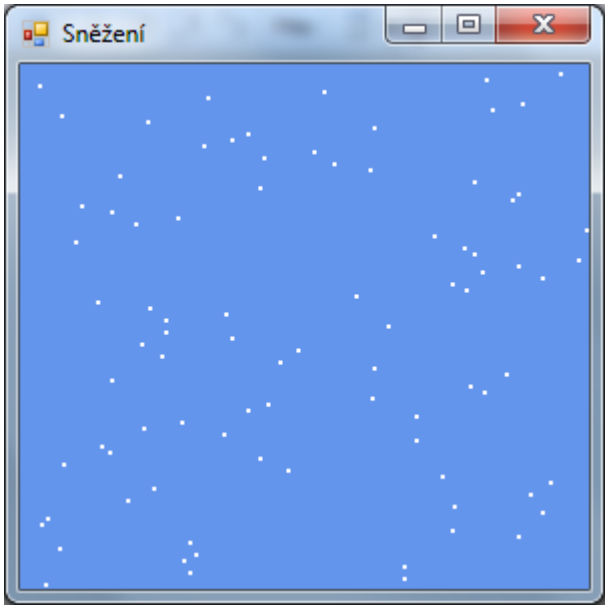
```
private void tlačítkoSmažVybrané_Click(object sender, EventArgs e)
{
    int index = SeznamJmen.SelectedIndex;
    if (index >= 0)
       SeznamJmen.Items.RemoveAt(index);
}

private void TlačítkoNahoru_Click(object sender, EventArgs e)
{
    int index = SeznamJmen.SelectedIndex;
    if (index > 0)
    {
        var záloha = SeznamJmen.Items[index];
        SeznamJmen.Items[index] = SeznamJmen.Items[index - 1];
        SeznamJmen.Items[index - 1] = záloha;
        SeznamJmen.SelectedIndex--;
    }
}

private void tlačítkoDolů_Click(object sender, EventArgs e)
{
    int index = SeznamJmen.SelectedIndex;
    if ((index >=0)&&(index<SeznamJmen.Items.Count-1))
    {
         var záloha = SeznamJmen.Items[index];
         SeznamJmen.Items[index] = SeznamJmen.Items[index + 1];
         SeznamJmen.Items[index + 1] = záloha;
         SeznamJmen.SelectedIndex++;
    }
}
```

Příklady na procvičení

1. Napište projekt **Stopy**. Do okna programu se budou vykreslovat stopy myši. Tam kde stiskneme myš, vykreslí se „myšinec“. Pro uchování bodů použijeme `List<Point>`.

2. Napište projekt **Sněžení**. Při tiku časovače se vykreslí náhodná vločka v horní části okna. Již vytvořené vločky se posunou o tři px dolů a náhodně o max. 2px vlevo nebo vpravo. Pro uchování vloček použijeme opět `List<Point>`.



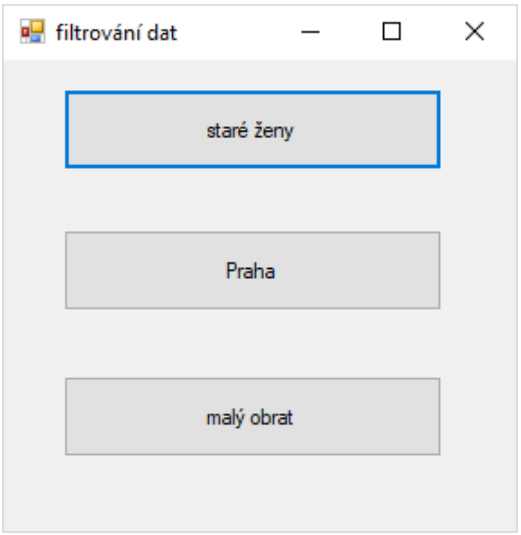


17. Zpracování tabulkových dat

Tabulky

Ke zpracování tabulkových dat budeme používat soubory ve formátu *.csv*. Jde o data exportovaná z tabulky v excelu do podoby textového souboru. Tabulka je přepsána po řádcích a jednotlivé buňky jsou oddělené středníkem.
Potřebné soubory naleznete v archivu ve složce **soubory k práci s tabulkovými daty**. Soubory budeme ukládat do složky příslušného projektu **bin/debug**,

Filtrování dat



Nejprve budeme pracovat se souborem **zakaznici.csv**.
Vytvoříme formulář projektu podle obrázku a nakopírujeme soubor **zakaznici.csv** do složky **bin/debug**.
Po stisku jednotlivých tlačítek se vytvoří soubor ve formátu *.csv*, do kterého se přepíše záhlaví tabulky a dále všechny řádky splňující daná kritéria.
Při stisku tlačítka „staré ženy“, vznikne soubor **stare_zeny.csv**. V něm budou ty záznamy, které splňují podmínku, že zákazník je žena narozená dříve než 1970.
Při stisku tlačítka „Praha“, vznikne soubor **se** záznamy, které splňují podmínku, že zákazník je z Prahy.
Při stisku tlačítka „malý obrat“, vznikne soubor se záznamy, které splňují podmínku, že zákazník má celkový obrat menší než 5000.

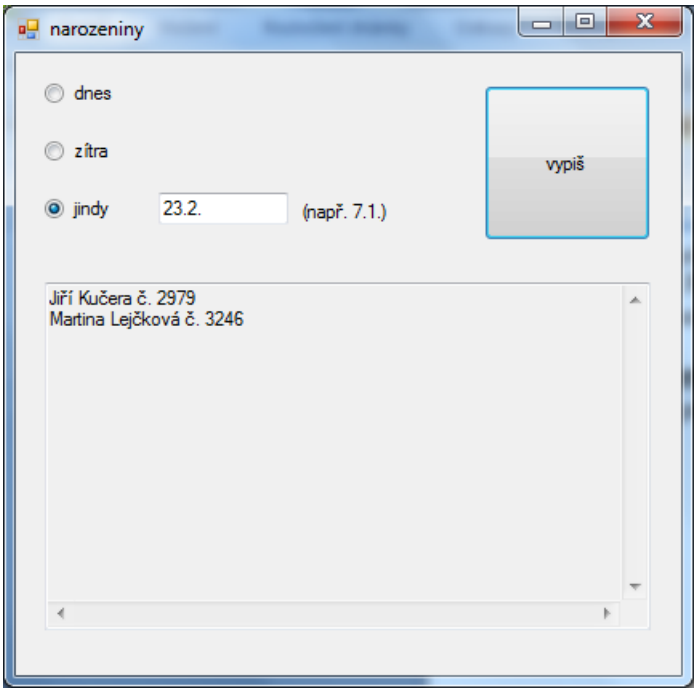
```
private void buttonStareZeny_Click(object sender, EventArgs e)
{
    //otevření souborů
    StreamReader zdroj = new StreamReader("zakaznici.csv",
    Encoding.Default);
    StreamWriter cil = new StreamWriter("stare_zeny.csv", false,
    Encoding.Default);
    //přepis záhlaví
    string zahlavi = zdroj.ReadLine();
    cil.WriteLine(zahlavi);
    //čtení a filtrování dat
    string radek;
    while ((radek= zdroj.ReadLine())!=null)
    {
        //z řádku vytvoříme pole
        string[] hodnoty = radek.Split(';');
        //výběr 4.hodnoty (pohlaví), osekání mezer
        //a převedení na malá písmena
        string pohlavi = hodnoty[3].Trim().ToLower();
        //výběr 5.hodnoty (rok), osekání mezer a převedení na celé číslo
        int rok = Convert.ToInt32(hodnoty[4].Trim());
        //filtrování a zápis
        if ((pohlavi == "žena") && (rok < 1970))
            cil.WriteLine(radek);
    }
        //uzavření souborů
        zdroj.Close();
        cil.Close();
        MessageBox.Show("hotovo");
}
```

Textový soubor budeme číst po řádcích. První řádek (hlavičku) přepíšeme beze změny. Další řádky zpracujeme pomocí metody `radek.Split(';')`. Metoda `Split` vytvoří z textového řetězce pole textových řetězců. Parametrem metody je znak oddělující jednotlivé položky. V našem případě je to ;. Znak je uveden v `''` nikoliv v uvozovkách. Je to datový typ znak (`char`), ne řetězec (`string`).

Protože pohlaví je ve čtvrtém sloupci tabulky, pracujeme se čtvrtou hodnotou pole, tj. s prvkem s indexem 3. Textový řetězec ještě upravíme metodami `Trim()` a `ToLower()`. Metoda `Trim()` odebere počáteční a koncové mezery. Pokud soubor vznikl z excelovské tabulky převodem do .csv, tak mezery nebudou, ale soubor lze editovat např. v poznámkovém bloku. Metoda `ToLower()` převede řetězec na malá písmena.

Rok narození je v pátém sloupci. Před převedením na číslo typu `integer` ho raději také opravíme metodou `Trim()`. Pak už jen ověříme patřičné podmínky a řádky, které je splňují přepíšeme do cílového souboru.

Práce s datem



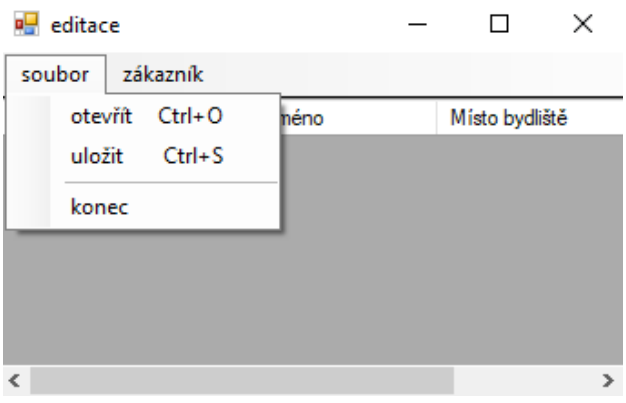
Tentokrát budeme používat soubor **zakaznici_s_dat_nar.csv**.
Tabulka má čtyři sloupce (číslo zákazníka, příjmení, jméno, datum narození).
Se zdrojovým souborem pracujeme stejně, tedy otevřeme pro čtení, první přečteme hlavičku (tentokrát ji zahodíme) a dál čteme po řádcích.

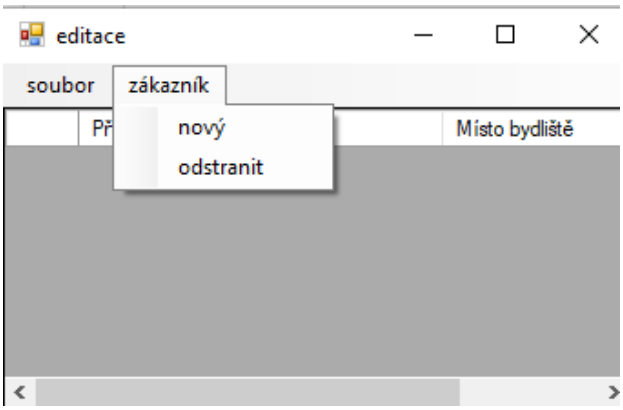
Přečtené řádky opět rozsekáme do pole metodou `Split(';')`. Do proměnné `datumTextem` přečteme poslední hodnotu v poli. Dále jí metodou `Convert.ToDateTime()` převedeme na instanci třídy `DateTime`, kterou pojmenujeme datum. Pak už stačí jen porovnat den a měsíc instance datum porovnat s hledaným datem.

Pro nastavení hledaného data podle požadavků nejprve vytvoříme instanci `DateTime dnes = DateTime.Today`. Je-li zaškrtnuto dnes, pak `hledane = dnes`. Je-li zaškrtnuto zítra, pak `hledane = dnes.AddDays(1)`. Metoda `AddDays(1)` přidá jeden den. Při zadaném datu se vytvoří nejprve textový řetězec z daného dne, měsíce a letošního roku. Ten se pak převede na instanci třídy třídy `DateTime` opět metodou `Convert.ToDateTime()`.

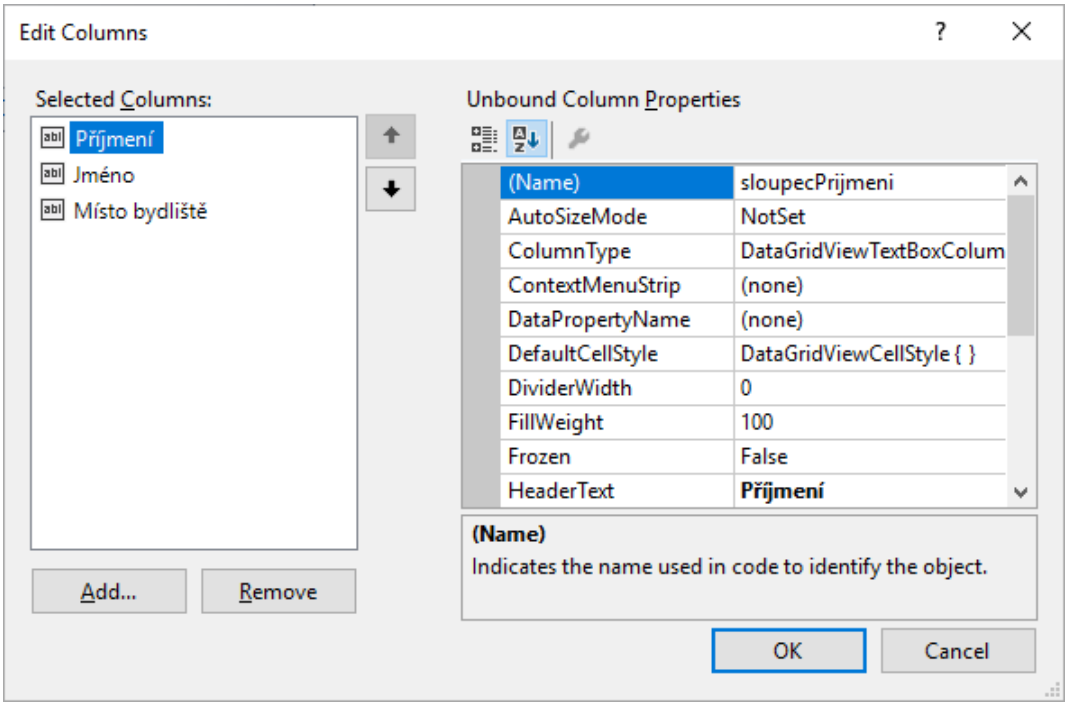
Editace dat

Budeme pracovat se souborem **zakaznici_editace.csv**.
V tomto projektu si zopakujeme práci s nabídkou. Použijeme ovládací prvek `MenuStrip`.





Pro práci s tabulkou použijeme ovládací prvek `DataGridView`. V tollboxu najdeme v kategorii Data.



Pro tabulku musíme nastavit vlastnosti jednotlivých sloupců. Počet a názvy sloupců přizpůsobíme souboru s daty. Zpracování dále budeme řešit obecně (jakože nevíme počet sloupců), abychom mohli přizpůsobit případným změnám ve zdrojovém souboru.

Takto vytvořená tabulka umožňuje přidávat na konec řádky a mazat řádky. Obě tyto akce, ale chceme řešit pomocí vlastních metod, proto je tabulce zakážeme. Pro tabulku nastavíme vlastnosti `AllowUserToAddRows` a `AllowUserToDaleteRows` na `False`. Pokud bychom nechali tabulku přidávat řádky, tak se nám po každém uzavření uloží na konec prázdný řádek.

```
using System;
using System.Text;
using System.Windows.Forms;
using System.IO;
namespace editace
{
    public partial class Form1 : Form
    {
        string hlavicka = null;
        public Form1()
        {
            InitializeComponent();
        }
        private void otevřítToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //otevření souboru a uložení hlavičky
            StreamReader soubor = new StreamReader("zakaznici_editace.csv",
            Encoding.Default);
            hlavicka = soubor.ReadLine();
            //vymazání tabulky
            tabulka.Rows.Clear();
            //načítání dat z tabulky
            string radek;
            while ((radek=soubor.ReadLine())!=null)

            {
                string[] hodnoty = radek.Split(';');
```

```
                tabulka.Rows.Add(hodnoty);
            }
            //zavření souboru
            soubor.Close();
        }

        private void konecToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }
        private void uložitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //otevření souboru a zápis hlavičky
            StreamWriter soubor = new StreamWriter("zakaznici_editace.csv",
           false, Encoding.Default);
            soubor.WriteLine(hlavicka);
            //zápis po řádcích
            int pocetRadku = tabulka.Rows.Count;
            int pocetSloupcu = tabulka.Columns.Count;
            for (int cisloRadku=0;cisloRadku<pocetRadku;cisloRadku++)
            {
                string radekZapisu = null;
                 for (int cisloSloupce=0;
                cisloSloupce<pocetSloupcu;cisloSloupce++)
                {
                    radekZapisu += tabulka[cisloSloupce, cisloRadku].Value;
                    if (cisloSloupce < pocetSloupcu - 1) radekZapisu += ";";
                }
                soubor.WriteLine(radekZapisu);
            }
            //zavření souboru
            soubor.Close();
        }
        private void novýToolStripMenuItem_Click(object sender, EventArgs e)
        {
            int pocetSloupcu = tabulka.Columns.Count;
            string[] prazdneHodnoty = new string[pocetSloupcu];
            for (int i = 0; i < pocetSloupcu; i++)
                prazdneHodnoty[i] = "";
            tabulka.Rows.Add(prazdneHodnoty);
        }
      private void odstranitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var aktivniBunka = tabulka.CurrentCell;
            if (aktivniBunka!=null)
            {
                int CisloRadku = aktivniBunka.RowIndex;
                tabulka.Rows.RemoveAt(CisloRadku);
            }
        }
    }
}
```

18. Třídy a objekty

Projekt zvířata

V okně formuláře se vykreslí obrázky tří zvířat. Každé zvíře umí vydat zvuk (textově), představit se jménem a poskočit (posunout se) doprava nahoru (o 30px v obou směrech).
Abychom každé zvíře nemuseli ovládat zvlášť, nadefinujeme si vlastní třídu *zvire.cs.*
Třídu vytvoříme v nabídce *Project ->add class*

```
namespace zvirata
{
    class zvire
    {
        public int X, Y;
        public Image obr;
        public string zvuk, jmeno;
        public string jmenujiSe()
        {
            string vystup = "Moje jméno je: " + jmeno;
            return vystup;
        }
        public void ozviSe(string z)
        {
            System.Windows.Forms.MessageBox.Show(z);
        }
        public void poskoc()
        {
            X += 30;
            Y -= 30;
        }
    }
}
```

Každý objekt ze třídy `zvire` bude mít číselné hodnoty `X,Y` (souřadnice levého horního rohu), `obr` (příslušný obrázek), textové hodnoty `zvuk` a `jméno`. Dále jsou pro něj definované tři metody. Metody `ozviSe(string z)` a `poskoc()` jsou bez návratové hodnoty. Metoda `jmenujiSe()` je s návratovou hodnotou. Metody `jmenujiSe()` a `poskoc()` jsou bez parametrů. Metoda `ozviSe(string z)` je s parametrem `z` (zvuk zvířete).

Zdrojový kód formuláře:

```
namespace zvirata
{
    public partial class Form1 : Form
    {
        zvire pes, kocka, kun, aktivni;
        public Form1()
        {
            InitializeComponent();
        }
        private void zvukToolStripMenuItem_Click(object sender, EventArgs e)
        {
            aktivni.ozviSe(aktivni.zvuk);
        }
        private void hopToolStripMenuItem_Click(object sender, EventArgs e)
        {
            aktivni.poskoc();
            Refresh();
        }
```

```
     private void pesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            aktivni = pes;
        }
        private void kočkaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            aktivni = kocka;
        }
        private void kůňToolStripMenuItem_Click(object sender, EventArgs e)
        {
            aktivni = kun;
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            pes = new zvire();
            pes.X = 100;
            pes.Y = 100;
            pes.obr = Properties.Resources.pes;
            pes.jmeno = "Alík";
            pes.zvuk = "HAF! HAF!";
            kocka = new zvire();
            kocka.X = 500;
            kocka.Y = 150;
            kocka.obr = Properties.Resources.kocka;
            kocka.jmeno = "Micka";
            kocka.zvuk = "MŇAU! MŇAU";
            kun = new zvire();
            kun.X = 200;
            kun.Y = 500;
            kun.obr = Properties.Resources.kun;
            kun.jmeno = "Otík";
            kun.zvuk = "ÍHAHA! ÍHAHA!";
            aktivni = pes;
        }

      private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics kp = e.Graphics;
            kp.DrawImage(pes.obr, pes.X, pes.Y);
            kp.DrawImage(kocka.obr, kocka.X, kocka.Y);
            kp.DrawImage(kun.obr, kun.X, kun.Y);
        }
     private void představSeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            string zprava = aktivni.jmenujiSe();
            MessageBox.Show(zprava);
        }
    }
}
```