Programování řízené událostmi (Event-driven programming)

přednáška NPRG031 Programování 2 25. 3. 2020

Programy řízené událostmi

Programy, které jsme psali dosud, vždy představovaly nějakou posloupnost příkazů.

Možná tam byly podmínky, cykly, funkce, ale za každým příkazem následoval (časově, ne nutně ve zdrojovém kódu) další příkaz a program tyto příkazy vykonával, dokud nedošel do konce.

Pokud potřeboval nějaký vstup od uživatele, zastavil se, blikal kurzorem a čekal, až uživatel zadá a on bude moci pokračovat.

Ale mnoho programů, které běžně používáme (v posledních cca 30 letech) se tak nechová.

Když spustíme vývojové prostředí nebo mailový klient nebo webový prohlížeč, tak se jen připraví k práci, většinou se nějak nakreslí – a čekají, co po nich budeme chtít.

Čekají, až přijde nějaká událost, kterou by mohly zpracovat.

Událost

Události známe z diskrétní simulace.

Můžeme mít složitý systém, který nemá nějaké ústřední velení, které by všechno řídilo, ale místo toho se skládá z mnoha částí, které jsou připravené na to, aby zpracovávaly události. A ta složitá funkčnost vzniká z toho, že každý dělá to, co umí.

Ústřední vedení je tam také, ale je směšně jednoduché.

V simulačních modelech to je smyčka, která vybírá z kalendáře události a nechává je zpracovat toho, koho se týkají.

V programech řízených událostmi je taková smyčka také, můžeme si ji představit jako

```
while (ještě_není_konec)
{
     Udalost u = PřijmiUdálost();
     ZpracujUdálost( u ) ;
}
```

Kdo zpracuje událost?

A v tom je právě to kouzlo. Ale postupně.

Úřad

Program řízený událostmi připomíná dobrý úřad.

Úředníci dobrého úřadu nechodí za člověkem domů, aby se ho ptali, jestli něco nepotřebuje (jako to dělají programy-neřízené-událostmi), ale sedí ve svém úřadu a čekají, až přijde nějaká událost.

Když přijdu na vrátnici finančního úřadu a obrátím se na vrátného, ten se mě zeptá, co chci a když odpovím „daň z příjmu fyzických osob", pošle mě do pátého patra.

V pátém patře by mohl být další vrátný, ale místo něj je tam jen nástěnka, která říká, že osoby s příjmením začínajícím A až D mají jít do jedněch dveří, osoby s příjmením E až J do dalších, takže se dozvím, že mám jít do dveří 504 k paní Melicharové. A paní Melicharová už zná moje daňová přiznání za posledních třicet let a je na ně specialista.

Počítač

Uživatel nemá moc možností, jak vyrobit událost.

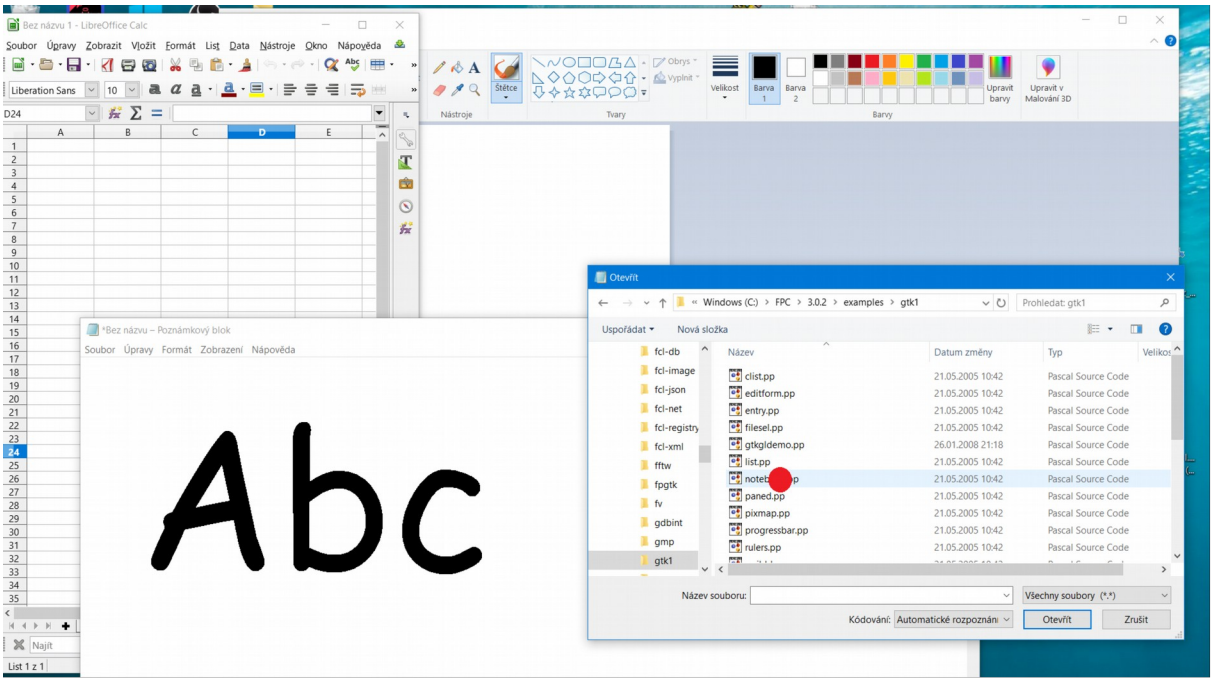
Konkrétně jsou to tyto možnosti:

- **klávesnice**
  - stisknout
  - pustit
- **myš**
  - stisknout
  - pustit
  - pohnout

Dlouho jsem se divil, co všechno dokážeme dělat s takhle skromnými prostředky… než přišly dotykové telefony, kde šmidláme jedním prstem.

Distribuce událostí

Představme si, že máme na svém počítači otevřeno několik oken, jako na obrázku



a zmáčkneme levé tlačítko myši v místě označeném červeným puntíkem.

Tím vytvoříme událost **stisk myši** (MouseDown) a její součástí je informace o tlačítku (**levé**) a souřadnicích (**1229, 750**).

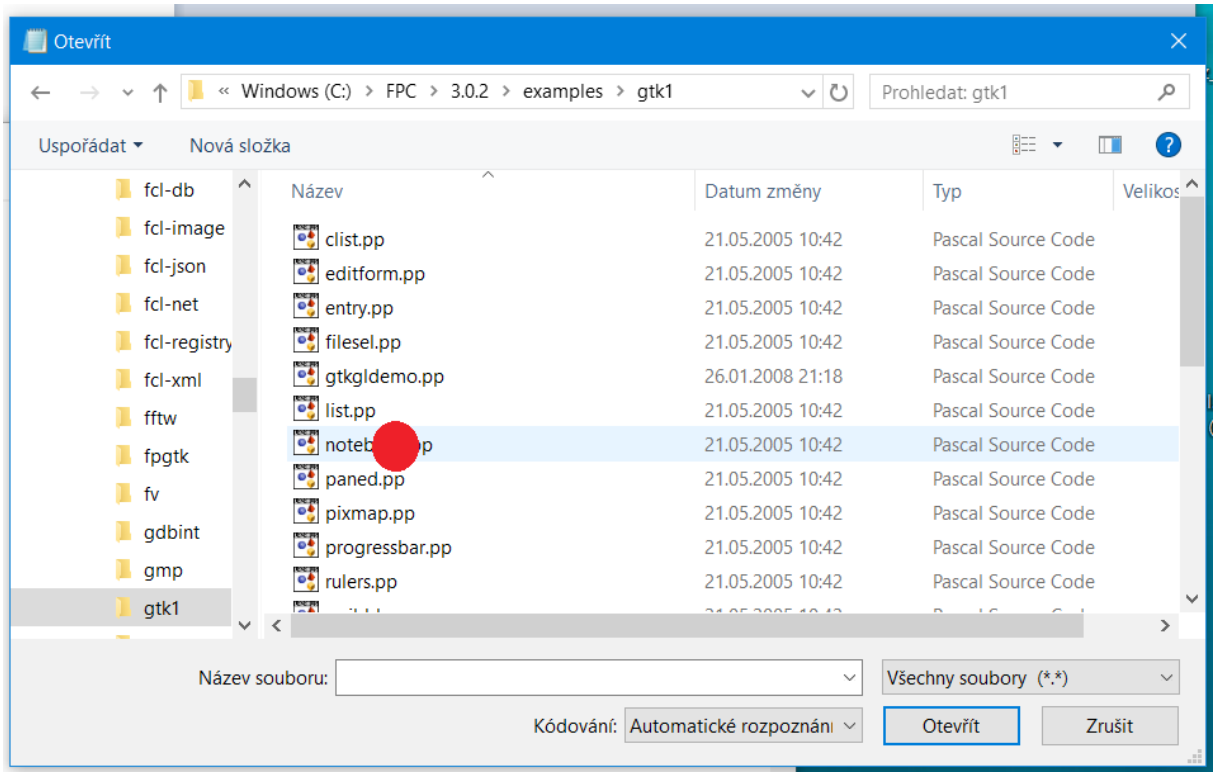
Tu událost by měl někdo zpracovat a nejsnazší je postupně se zeptat všech. Takže se můžeme postupně obracet na všechna okna a každé z nich požádat, aby zpracovalo tuto událost.
Trik: Je dobré se ptát v pořadí odpředu (**Z-order**), abychom se v případě, že se okna překrývají, dříve zeptali toho, které je navrchu.

Některá okna odpoví, že tu událost zpracovat nemohou, protože je jinde, než jsou ona. Pokud událost nezpracuje nikdo, zůstane nezpracovaná, někdy se nám to ohlásí pípnutím, někdy se nestane nic.

Hierarchie

Když událost dorazí k oknu, kterého by se mohla týkat, okno ji zkusí zpracovat úplně stejným způsobem – bude se postupně ptát všech svých podřízených, jestli dokážou tu událost zpracovat.

Podřízení – to jsou ta tlačítka, textová pole, combo-boxy a další stavební prvky okenního (nejen Windows) systému. Budeme jim říkat **komponenty**.



Takhle událost putuje, jako klient v úřadu, až dorazí ke komponentě, která je specializovaná na jednu činnost, jako dotyčná paní na daňovém úřadě.

Komponenta

V tomto případě je to komponenta nazývaná **ListBox**, která slouží k zobrazování a ovládání seznamů.

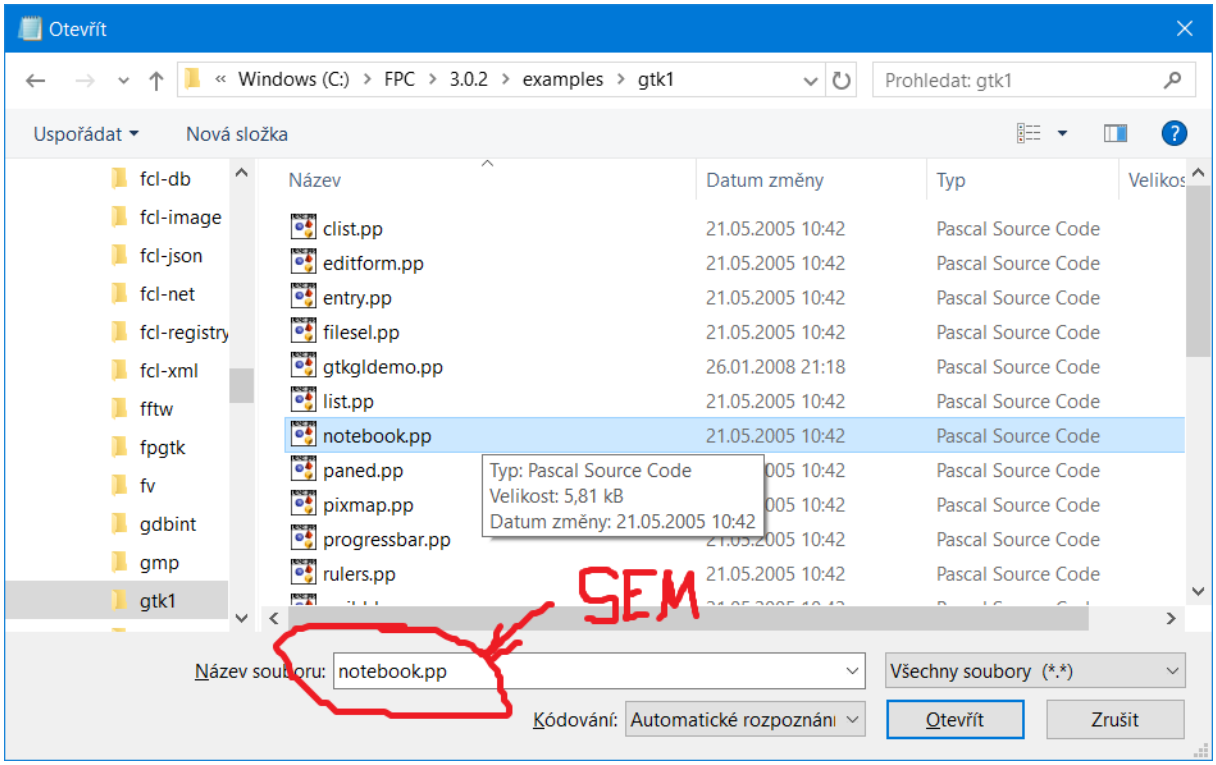
Má seznam položek, které obsahuje, umí je vykreslit různými způsoby, ví, jak je vysoký jeden řádek a široký jeden sloupec…

…takže když přijde událost **stisk myši** na určité souřadnici, komponenta ListBox si umí spočítat, na kterou položku uživatel ukázal a zareaguje na to tak, že ji zvýrazní.

Transformace událostí

Tím, že komponenta ListBox rozumí své práci, mohla převést nízkoúrovňovou událost (**na jedné z milionu souřadnic bylo zmáčknuté tlačítko myši**) na událost vyšší úrovně (**byla vybraná položka ze seznamu**). Podobně jako když archeolog v prohlubních na kameni rozpozná písmo.

Komponenta navíc může být připravena na to, že když se něco takového stane, má provést nějakou akci (zavolat nějakou funkci). V tomto případě, nevím, jestli jste si toho všimli, výběr položky ze seznamu souborů způsobí, že se jméno vybraného souboru napíše do políčka pro název souboru, viz obrázek.



Polymorfismus

Pokud by vám přišlo divné, jak může mít okno **seznam podřízených**, z nichž každý je **jiného typu** (ale po zkušenosti s jazykem Python vám to asi divné nepřijde), tak je to proto, že všichni ti podřízení jsou odvození od jednoho společného předka a přepisují (override) metodu (musí být **virtuální**) na zpracování události, takže nadřízený může každému svému podřízenému říci, aby zpracoval událost, aniž by se staral o to, jakého typu ten podřízený je. Polymorfismus v praxi.

Programy řízené událostmi

Programy řízené události potom vytváříme tak, že z připravených komponent (ty už dostaneme hotové, i když si někdy také můžeme vytvořit vlastní) skládáme hierarchii (ten úřad, kterým budou putovat události) a navíc vytváříme funkce, které se mají zavolat v případě, že nastane nějaká konkrétní událost u určité konkrétní komponenty.

Výsledný program je tedy jednak ta provázaná struktura komponent a oken a jednak řada funkcí reagujících na jednotlivé události.

Je tam také nějaký „hlavní program", ale ten nebývá zajímavý, protože po počáteční inicializaci jen provádí smyčku, která zpracovává události.

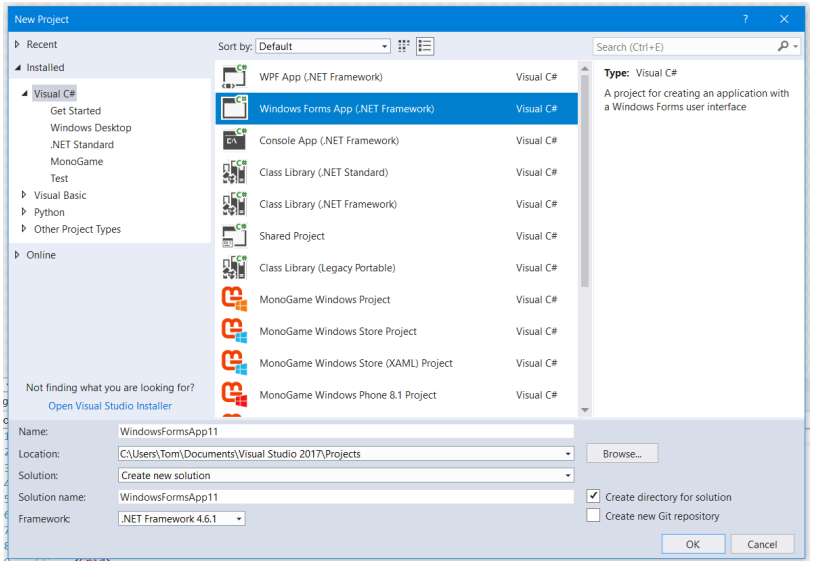
Ladění

Když je hlavní program cyklus, který zpracovává události, nemá velký smysl hledat chyby krokováním celého programu. Takže pokud něco nefunguje, hlavní nástroj je umístit si **breakpoint** do funkce, která by se měla volat, podívat se nejdříve, jestli se vůbec volá (jestli se program na breakpointu zastaví nebo ne) a pokud ano, tak teprve od tohoto místa krokovat.

Jak na to ve Visual Studiu

Vytvoření aplikace

Začneme tím, že si vytvoříme novou aplikaci a tentkrát si vybereme typ **Windows Forms App.** Budu ukazovat Windows Forms App, ale pokud nepracujete pod Windows a vyberete si místo toho **WPF App**, bude to podobné. Výběrem typu aplikace zároveň říkáme, jakou knihovnu ovládacích prvků/komponent (obojí je nepřesné) budeme používat.



První okenní aplikace

Pokud už dál potom nebudeme nic měnit a jen zmáčkneme **F5**, vytvořili jsme náš první okenní program. Měli bychom vidět prázdné okno a může nám přijít trochu nezajímavé, na druhou stranu, takhle aplikace dovede

- přesouvat okno myší
- minimalizovat, maximalizovat, měnit velikost
- zobrazovat svoji ikonku na liště mezi běžícími programy
- zobrazovat se v seznamu běžících programů
- dělit se o procesor s ostatními běžícími programy

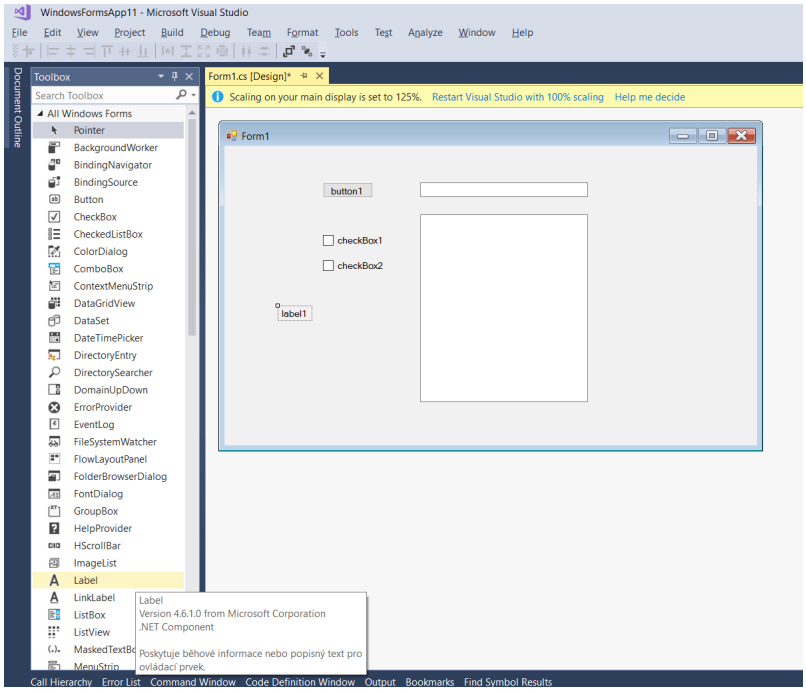
...a vůbec řadu věcí, o které se většinou nechceme starat sami. Takže to není malý úspěch!

Návrhář a komponenty

Pokud se po spuštení první aplikace vrátíme do Visual Studia, uvidíme okno **Návrháře (Design)**, které odpovídá vzhledu našeho okna. Můžeme myší změnit jeho velikost (nezkoušejte ho posunout, nejde to) a když si myší nebo zkratkou (**Ctrl-Alt-X**) zobrazíme okno **Toolbox**, můžeme naše okno osázet komponentami.

Okno Toolbox si můžete „špendlíkem" připíchnout, aby se neschovávalo.

Komponenty jsou v okně Toolbox rozděleny do různých kategorií, pokud nevíte, ve které kategorii hledat, tak ve skupině **All Windows Forms** jsou všechny.



Vlastnosti

Každá z komponent má řadu vlastností určujících její vzhled i chování, to platí i pro samotné okno.

Vlastnosti můžete prohlížet a nastavovat v okně **Properties**, které zobrazíte myší nebo klávesovou zkratkou **Alt-Enter.**

Vlastnosti komponent (a oken) jsou opravdu properties, tedy něco, co vypadá a používá se jako proměnná (datová složka objektu), ale ve skutečnosti je to dvojice funkcí, jedna pro čtení a druhá pro zápis.

Takže je možné jednak hlídat, jaké hodnoty se jednotlivým vlastnostem pokoušíme přiřadit a jednak toto přiřazení může mít vedlejší efekt, například když změníme vlastnost **Text** komponenty typu **Label,** změní se také její vlastnost **Size,** pokud jsme nezměnili její vlastnost **Autosize** z výchozí hodnoty **true** na **false.** Zároveň se díky tomu změna hodnoty vlastnosti ihned promítne do zobrazení komponenty v okně Návrháře.

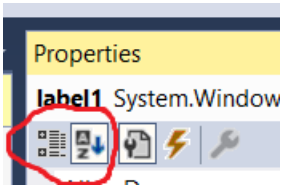
Některé vlastnosti sestávají z více částí, mají nalevo tlačítko [+].

Některé vlastnosti dovolují výběr hodnoty ze seznamu (třeba vlastnost s hodnotami typu **Bool.**

A některé vlastnosti (třeba **Font**) mají napravo u hodnoty ikonku […] naznačující, že na nastavování této vlastnosti můžeme vyvolat zvláštní okno.

Vlastnost (Name)

Vlastnosti jsou řazené podle kategorií nebo podle abecedy, přepíná se to ikonkami v záhlaví okna



Každá komponenta má v okně svou proměnnou, abychom nemuseli přemýšlet nad jmény, dokud nás nezajímají, tak jsou automaticky pojmenovávány jako **TYP** a **pořadové číslo.** Tedy například **label1.**

Když nás jméno této proměnné začne zajímat, tak ji v okně Properties můžeme změnit a aby bylo v seznamu vlastností (podle abecedy) na začátku, najdeme ji tam jako vlastnost **(Name)** (ano, se závorkami).

Obvyklé vlastnosti

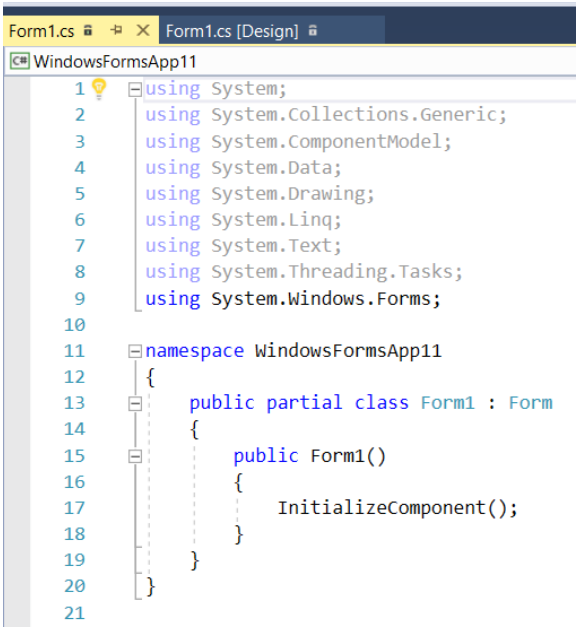
Vlastnosti, které má skoro každá komponenta, jsou **Left, Top, Width, Height, Visible, Enabled, Text**, význam odhadněte.

Dvojí zdrojový kód

Když své okno zaplníme komponentami a program spustíme (skoro kdykoliv můžeme program spustit), měl by už vypadat jako opravdový program… A ty komponenty jsou opravdu stavební prvky Windows, žádná laciná napodobenina, takže třeba editační políčko **TextBox** umí psát, mazat, přepínat režim vkládání a přepisování, ale třeba také kopírovat do a ze schránky.

Takže když vyrobíme takový dobře vypadající program mohlo by nám začít vrtat hlavou, **kde je zdrojový kód?**

Zdrojový kód zobrazíme pomocí myši nebo klávesovou zkratkou **F7** nebo **Ctrl-Alt-0**, podle verse a nastavení Visual Studia…



...a uvidíme, že tam není vůbec nic zajímavého! Naše okno (třída **Form1**) je odvozené od třídy **Form** a jenom přepisuje konstruktor. Nic víc!

Záhadu objasní klíčové slovo **partial** v definici třídy, které říká, že tohle je možná jenom část definice třídy a pokračování může (ale nemusí) být v nějakém jiném souboru nebo souborech.

Každé okno má standardně zdrojový kód ve dvou souborech, v prvním jsou ty zajímavé věci související s chováním (zatím jsme žádné nenapsali) a ve druhém hodnoty vlastností – to, co jsme naklikali myší a nastavili v okně Properties.

Druhý zdrojový soubor zobrazíme třeba pomocí **Průzkumníka řešení (Solution Explorer)** nebo tak, že se klávesovou zkratkou **F12** podíváme na definici funkce **InitializeComponent().**

```
22
23      #region Windows Form Designer generated code
24
25      /// <summary>
26      /// Required method for Designer support - do not modify
27      /// the contents of this method with the code editor.
28      /// </summary>
29      private void InitializeComponent()
30      {
31          this.button1 = new System.Windows.Forms.Button();
32          this.checkBox1 = new System.Windows.Forms.CheckBox();
33          this.checkBox2 = new System.Windows.Forms.CheckBox();
34          this.textBox1 = new System.Windows.Forms.TextBox();
35          this.textBox2 = new System.Windows.Forms.TextBox();
36          this.label1 = new System.Windows.Forms.Label();
37          this.SuspendLayout();
38          //
39          // button1
40          //
41          this.button1.Location = new System.Drawing.Point(148, 55);
42          this.button1.Name = "button1";
43          this.button1.Size = new System.Drawing.Size(75, 23);
44          this.button1.TabIndex = 0;
45          this.button1.Text = "button1";
46          this.button1.UseVisualStyleBackColor = true;
47          //
48          // checkBox1
49          //
50          this.checkBox1.AutoSize = true;
51          this.checkBox1.Location = new System.Drawing.Point(148, 132);
52          this.checkBox1.Name = "checkBox1";
53          this.checkBox1.Size = new System.Drawing.Size(98, 21);
```

Nesahat!

Definice této funkce se nachází v regionu nadepsaném

```
#region Windows Form Designer generated code
```

, což by se dalo přeložit zhruba jako

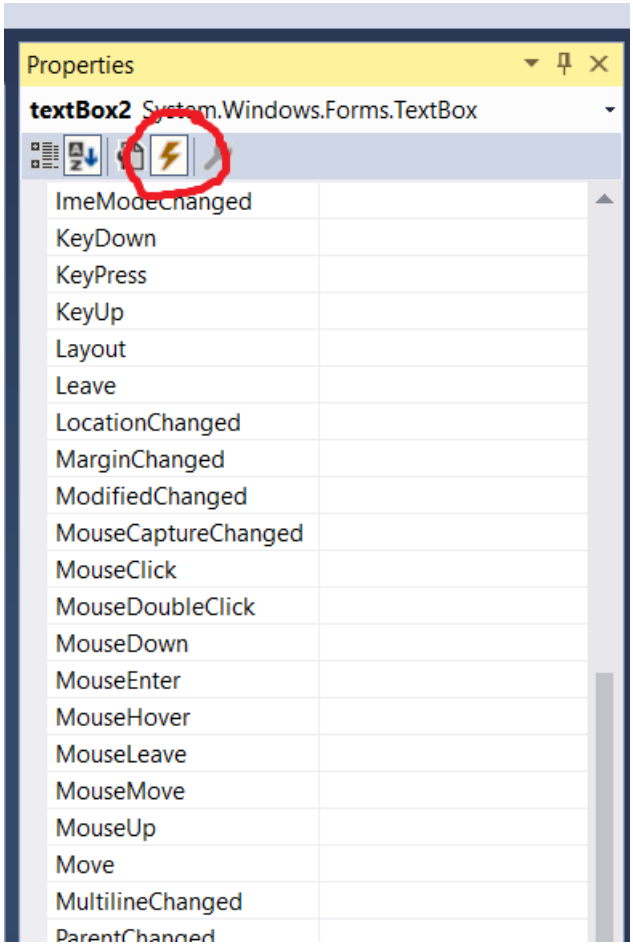
„Sem se ani nedívej a v žádném případě nic neměň!“

Tuto část zdrojového kódu **píše** Visual Studio podle toho, co děláme v okně Návrháře a také ji po sobě čte. Takže pokud tu něco pokazíme, je pravděpodobné, že tuto část nedokáže přečíst a okno Návrháře a přilehlé nástroje nám nezobrazí.

Události

Když chceme (konečně) napsat nějaký kód, budeme to dělat tak, že jednotlivým komponentám (nebo i oknu) přiřadíme nějakou funkci, která se má zavolat, pokud dojde k určité události.

Slouží k tomu opět okno **Properties,** jen se ikonkou v záhlaví přepneme ze zobrazení vlastností na zobrazení událostí (což jsou také vlastnosti, jenom jejich hodnotou nejsou čísla a stringy, ale funkce):



Pokud chceme reagovat třeba na událost **MouseMove,** nemusíme vytvářet funkci, stačí zaklikat (double click) do (prázdného) políčka napravo od jména události a Visual Studio takovou funkci vytvoří, dá jí výchozí jméno složené ze jména komponenty a jména události – a přepne nás do zdrojového kódu:

```
private void textBox2_MouseMove(object sender, MouseEventArgs e)
{

}
```

Nemazat!

Takto vytvořené funkce v žádném případě nemažte, ze stejného důvodu, z jakého nemáte editovat generovanou část zdrojového kódu. Pokud je chcete přejmenovat, tak použijte přejmenovávání **Ctrl-R-R** nebo při

vytváření místo dvojitého kliknutí napište požadované jméno funkce apotvrďte **Enter.**

K tomu pohybu myši

Když už tu funkci máme, napíšeme do ní následující kód:

```
19
20          private void textBox2_MouseMove(object sender, MouseEventArgs e)
21          {
22              Random rnd = new Random();
23              textBox2.Left = rnd.Next(Width - 20);
24              textBox2.Top = rnd.Next(Height - 30);
25          }
26      }
27  }
```

...a máme program, kde textové políčko utíká před myší.

Typická událost

Viděli jsme, že taková komponenta má desítky událostí, na které je připravená reagovat. Přitom nejčastější událost, na kterou budeme chtít reagovat třeba u tlačítka, bude událost **Click** (to znamená **MouseDown** a potom **MouseUp**). Když chceme založit funkci pro obsluhu typické události, nemusíme používat okno Properties, stačí přímo v okně Návrháře na komponentu poklikat.

Vlastnosti své a cizí

Komponenty patří oknu, všechny funkce pro obsluhu událostí také, takže kterákoliv taková funkce může **nastavovat vlastnosti své i ostatních komponent** (a okna).

Například

```
27          private void button1_Click(object sender, EventArgs e)
28          {
29              if (checkBox1.Checked)
30              {
31                  textBox1.Text = "zaškrtnuto";
32              }
33          }
```

Sdílení funkcí na obsluhu událostí

Údaj o tom, která funkce má obsluhovat kterou událost, je jen **odkaz** na funkci, takže nic nebrání tomu, aby několik komponent volalo stejnou funkci nebo aby jedna komponenta volala stejnou funkci pro několik událostí, stačí jen v okně Vlastností tu kterou funkci místo zakládání **vybrat.**

Pozor: funkce pro obsluhu různých událostí můžou mít různé parametry (třeba funkce pro obsluhu události **MouseDown** dostane údaje o poloze myši, událost **Click** ne (kliknout lze i mezerníkem)), takže se pro výběr nabízejí jen funkce správného typu.

Parametr sender

Všechny funkce pro obsluhu událostí mají parametr pojmenovaný **sender** – komponenta, od které přišla událost. Díky tomu se uvnitř takové společné funkce můžeme rozhodnout podle toho, kdo událost poslal nebo přímo pracovat s parametry daného objektu.

Jenom… parametr sender je typu **object**, pokud funkci volá třeba tlačítko a my chceme pracovat s jeho vlastností **Text,** musíme parametr sender **přetypovat:**

```
27          private void button1_Click(object sender, EventArgs e)
28          {
29              ((Button)sender).Text = "KLIK";
30          }
```

Dotaz na typ

Pomocí operátoru **is** se také můžeme zeptat, jakého typu je daný objekt:

```
27          private void button1_Click(object sender, EventArgs e)
28          {
29              if (sender is Button)
30              {
31                  ((Button)sender).Text = "KLIK";
32              }
33              else
34                  MessageBox.Show("Tohle není tlačítko!");
35          }
```

Operátor **is** vrací **true**, pokud objekt je stejného typu nebo typu odvozeného (takže nějaké vylepšené tlačítko by prošlo také).

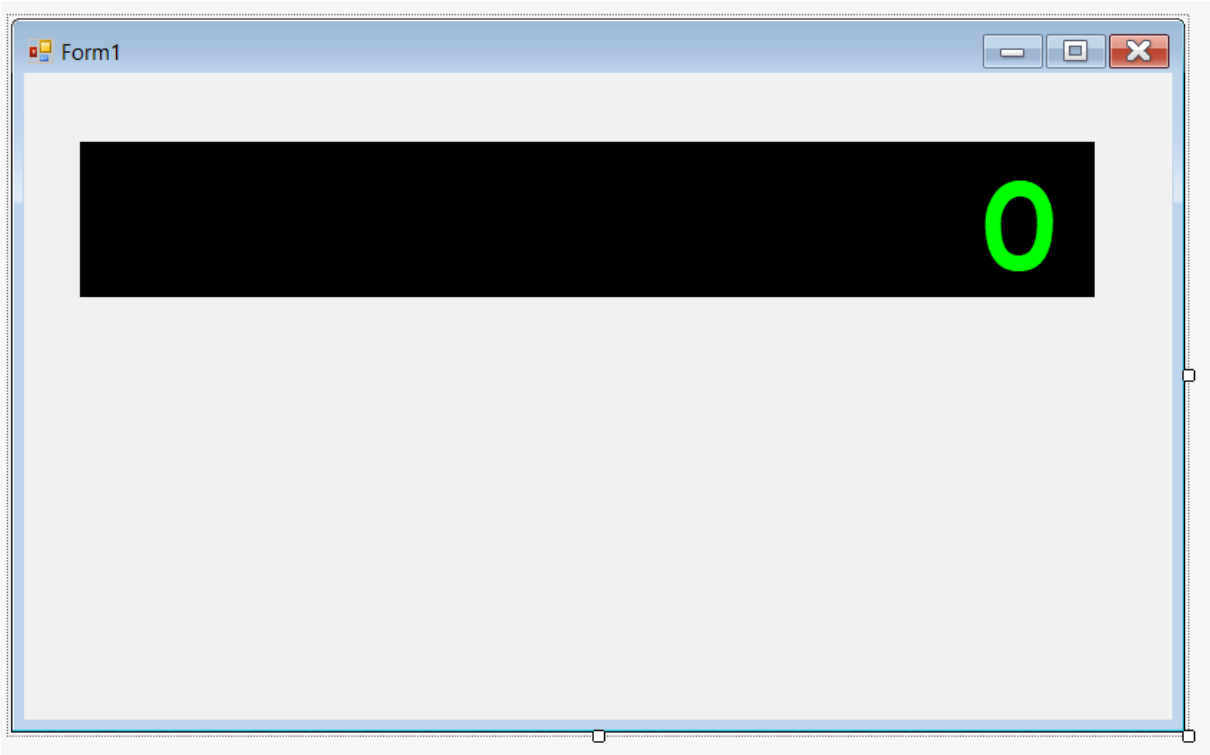
Příklad – kalkulačka

Vyrobme si kalkulačku, čtyři základní operace, bez priorit.

Vzhled

Začneme grafickým návrhem, nejdříve displej a potom tlačítka.

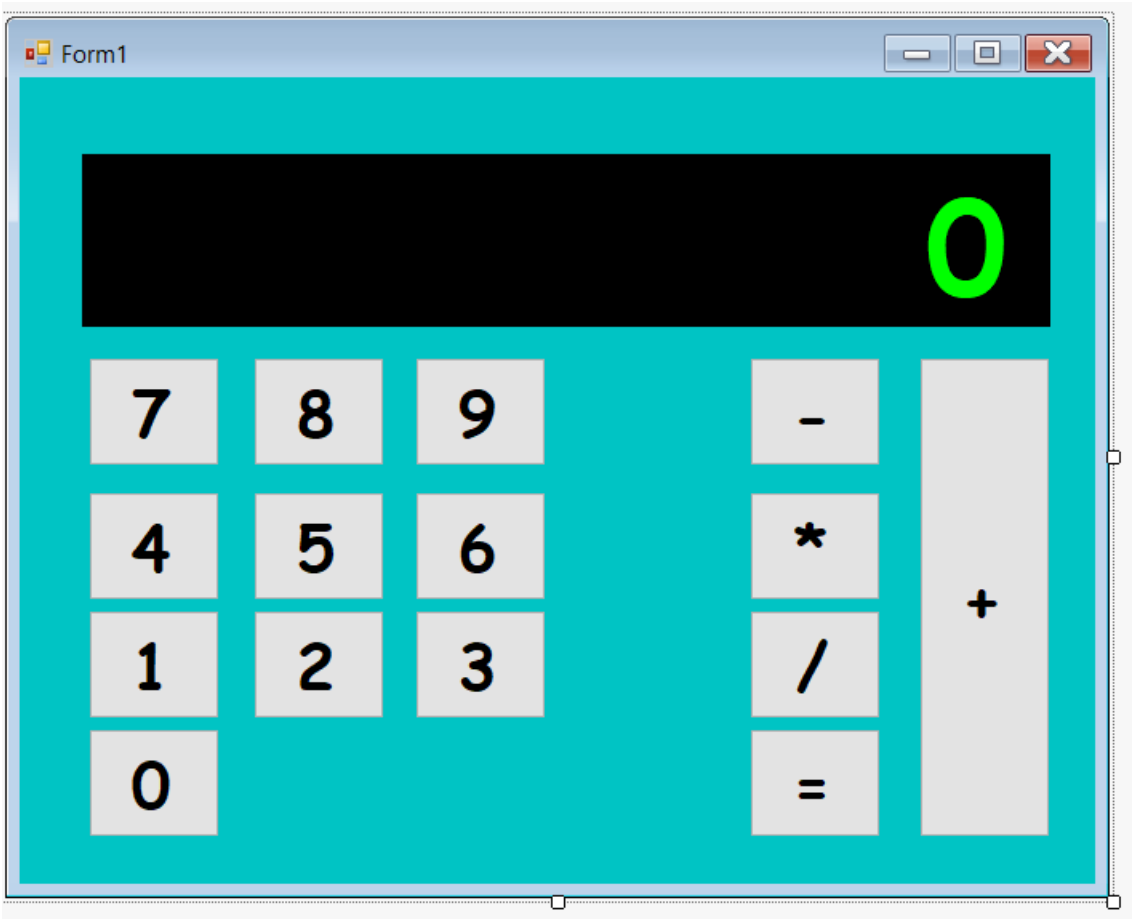
Displej bude komponenta typu **Label**, budeme se na ni odkazovat ve zdrojovém kódu, proto jí změníme jméno na **D** (tlačítka přejmenovávat nebudeme) a nastavíme jí vlastnosti **AutoSize** (aby se nesmrskával podle velikosti textu), **Text, TextAlign, Font, ForeColor** a **BackColor** a myší v okně návrháře nastavíme velikost:



Pak přidáme tlačítka (komponenta **Button**), u toho můžeme využít kopírovaní, takže si vyrobíme jedno tlačítko, nastavíme mu požadovaný vzhled (vlastnost **Font** a velikost) a pak ho rozkopírujeme do sloupečku a pak zkopírujeme celé sloupečky (kopírování je častý zdroj chyb!).

Trik: Když si v okně Properties vyberete nějakou vlastnost, potom v okně Návrháře po výběru komponenty můžete začít psát a Visual Studio vás automaticky přepne do okna Properties. Neboli ta tlačítka se dají popsat docela rychle – myší na ně ukazujete a druhou rukou píšete číslice a operace.

Nakonec změníme vlastnost **BackColor** samotného okna:



Vizuální návrh pomocí komponent a myši nám dovoluje rychle vytvořit **prototyp aplikace**, u kterého si můžeme ujasnit, jestli to je to, co po nás zákazník chce.

Funkčnost

Na to, aby kalkulačka počítala, nám stačí dvě funkce.

Klávesa s číslicí

Připíše svou číslici na konec displeje, předtím smaže displej, pokud to je potřeba.

Klávesa s operací

Podívá se, jestli nemá uloženou operaci od minula, pokud ano, tak ji vyhodnotí s **uloženou hodnotou** a aktuální hodnotou na displeji, zapíše výsledek na displej, uloží jako minulou hodnotu a (teprve teď) operaci ze stisknutého tlačítka uloží jako **minulou operaci** a nastaví, že se **má smazat displej.**

K tomu potřebujeme, aby si kalkulačka pamatovala zmíněné údaje: **jestli se má smazat displej**, **minulou hodnotu** a **minulou operaci.**

Operace **=** bude dělat to samé (volat stejnou funkci) jako ostatní operace.

Zdrojový kód

Celý přidaný zdrojový kód tedy vypadá takto:

```
19          bool mamSMazatDisplej = true;
20          int minulaHodnota;
21          char minulaOperace = '?';
22          private void Cislice_KLIK(object sender, EventArgs e)
23          {
24              if (mamSMazatDisplej)
25              {
26                  D.Text = "";
27                  mamSMazatDisplej = false;
28              }
29              D.Text += ((Button)sender).Text;
30          }
31
32          private void Operace_KLIK(object sender, EventArgs e)
33          {
34              int x = Int32.Parse(D.Text);
35              switch (minulaOperace)
36              {
37                  case '+': x = minulaHodnota + x; break;
38                  case '-': x = minulaHodnota - x; break;
39                  case '*': x = minulaHodnota * x; break;
40                  case '/': x = minulaHodnota / x; break;
41              }
42              D.Text = x.ToString();
43              mamSMazatDisplej = true;
44              minulaHodnota = x;
45              minulaOperace = ((Button)sender).Text[0];
46          }
```

...jednoduché, ne?

Teď už víte dost

Teď už víte dost k tomu, abyste si třeba ke svému zápočtovému programu ze zimního semestru dokázali napsat okenní rozhraní (pokud už jste si ho nenapsali v Pythonu, samozřejmě).

Tak běžte a zkoušejte, hrajte si a ptejte se.

Odpovědi na případné otázky budou zase na mé stránce: https://ksvi.mff.cuni.cz/~holan/py/otazky_200325.html , dotazy můžete posílat mailem nebo anonymně pomocí formuláře na zmíněné stránce (aspoň pípněte, že jste dočetli až sem).

Hodně zdaru!