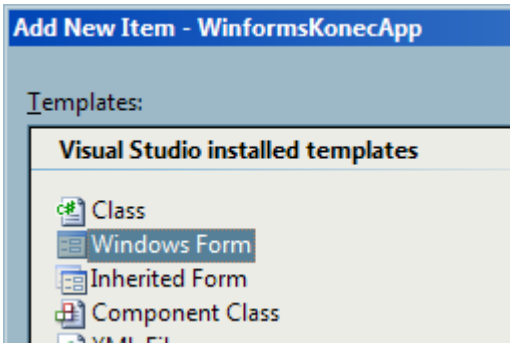
Dialogy

- pro komunikaci s uživatelem – nastavení většího množství hodnot najednou
  - hostí ovládací prvky
- speciální případ okna (z pohledu OS Windows) – v .NET shodné s běžným formulářem
- základní vlastnosti:
  - zpravidla neměnná velikost
  - implicitní reakce na klávesy (ENTER, ESC)
  - modální x nemodální dialog
- chování vůči vlastníkovi:
  - aplikačně modální – zákaz vlastníka (většina dialogů)
  - systémově modální – vždy na vrchu
  - úkolově modální (task modal) – zákaz všech top-level oken daného vlákna

Vlastní dialogy

Vytvoření

- v .NET shodné s běžným formulářem
- možnosti
  - programově
    ```
    Form dialog = new Form();
    ```
    - vzhled nepůjde vytvořit v návrháři
  - přes návrhář VS2005 – menu Project→Add new Item



- Vzhled okna – obvykle neměnná velikost, nezobrazuje se v liště úloh

```
dialog.FormBorderStyle = FormBorderStyle.FixedDialog;
dialog.MaximizeBox = false;
dialog.MinimizeBox = false;
dialog.ShowInTaskbar = false;
```

- spuštění (zobrazení):
  - modální: `dialog.Show()`
  - nemodální: `dialog.ShowModal()`

Přidání ovládacích prvků

- jasné, viz cvičení

Kontrola zadaných hodnot

- „je ta hodnota > 0?“, „je to telefonní číslo?“
- ruční – při zavření dialogu
  - něco špatně = stornování zavření dialogu (událost `Form.Closing`) – viz min. přednáška
- automatická – událost `Validating`
  - zděděna z `Control`
  - vyvolá se při opuštění prvku (ztrátě fokusu)
- podrobněji viz cvičení

Vstup a výstup dat

- viz cvičení

Ukončení dialogu

- Obvyklé možnosti:
  - Stiskem OK – uživatel chce nastavené hodnoty použít (provést akci)
  - Stiskem Storno – nechce
  - Něco jiného (velmi neobvyklé)
- Nutno přidat tlačítka „OK“ a „Storno“
- Rozeznání způsobu ukončení – metoda `ShowModal()` vrací `DialogResult`

```
enum DialogResult
{
	//		Tlačítka:
	Abort,		// Zrušit
	Cancel,		// Storno
	Ignore,		// Ignorovat
	No,		// Ne – u MessageBox
	None,
	OK,		// OK
	Retry,		// Znovu
	Yes		// Ano – u MessageBox
}
```

- Třída `Form` – Vlastnost `DialogResult DialogResult` – před ukončením naplnit podle tlačítka (OK/Storno)
  - Defaultní hodnota `None`

```
private void buttonOK_Click(object sender, EventArgs e)
{
	this.DialogResult = DialogResult.OK;
	this.Close();
		// implicitne nastavi DialogResult na DialogResult.Cancel
	}

private void buttonStorno_Click(object sender, EventArgs e)
{
	this.Close();
```

```
}
```

- V nadřazeném formuláři kontrola typu ukončení a akce:

```
private void buttonNastavData_Click(object sender, EventArgs e)
{
        NastavDataDialog dialog = new NastavDataDialog();
        DialogResult dr = dialog.ShowDialog();
        if (dr == DialogResult.OK)
        {       // Akce, vyzvednutí dat
                label1.Text = "OK";
        }
        else
        {       // obvykle nic neděláme
                label1.Text = "Storno";
        }
}
```

- Nedostatky ručního nastavení `DialogResult` v dialogu dle tlačítka – nefunguje výchozí chování při stisku kláves ENTER (= tlačítko OK) a ESC (= tlačítko Cancel)
- Řešení – `Form` Vlastnosti:
  - `AcceptButton` – přiřadit referenci na tlačítko OK
  - `CancelButton` – na Cancel

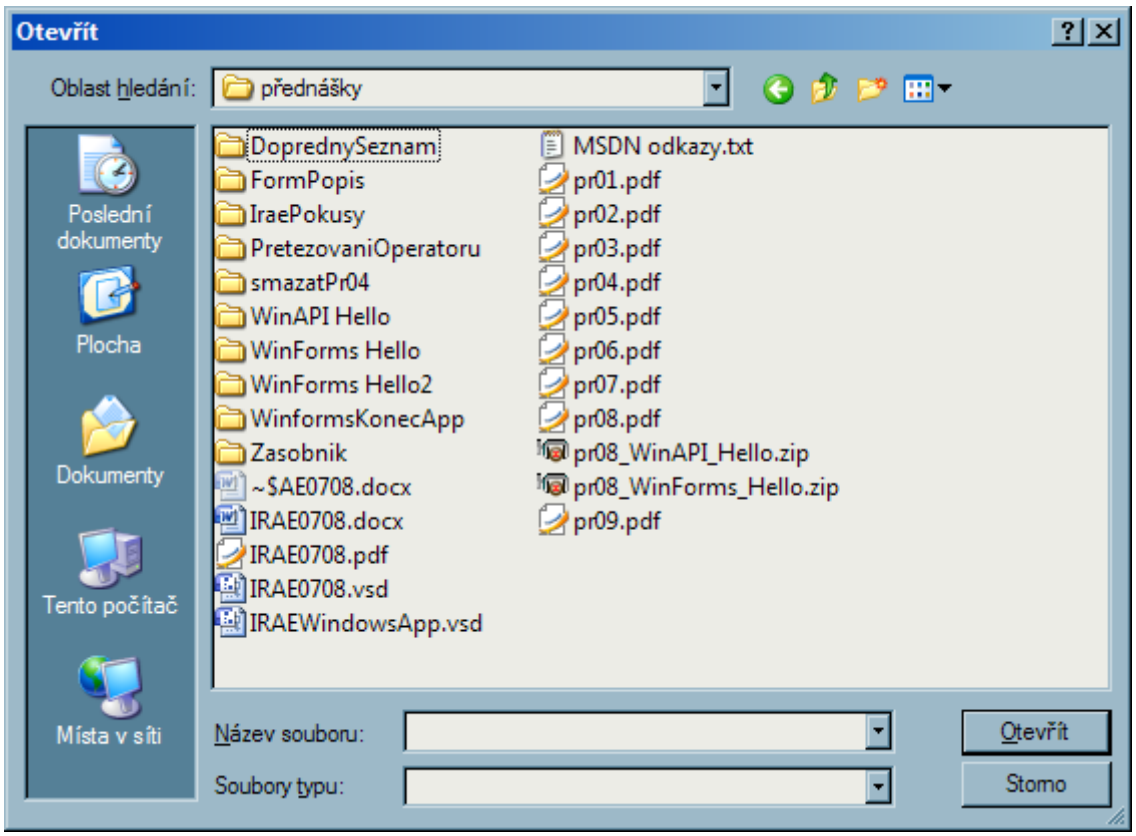
```
public NastavDataDialog()
{
        InitializeComponent();
        this.AcceptButton = this.buttonOK;
        this.CancelButton = this.buttonStorno;
}
```

    - lze udělat pohodlně i v návrháři
    - v tomto případě události tlačítek není nutné vůbec obsluhovat – tlačítka získají automaticky požadovanou funkčnost

Běžné dialogy

- uživatel je zná a očekává
- v .NET předpřipraveny jako třídy
- princip práce – stejný jako u normálního dialogu:
  - pouze není nutno vytvářet
- Dialogy:
  - `ColorDialog`
  - `OpenFileDialog`
  - `SaveFileDialog`
  - `FolderBrowserDialog`
  - `FontDialog`
  - `PageSetupDialog`
  - `PrintDialog`
  - `PrintPreviewDialog`

Příklad – OpenFileDialog



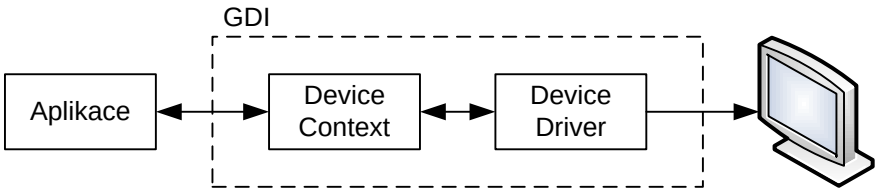
- Metody
  - `DialogResult ShowDialog()` zobrazí dialog
- Vlastnosti:
  - `string FileName` (Get, Set) – cesta + jméno souboru („Název souboru:“)
  - `string Filter` (Get, Set) – filtr pro výběr souborů („Soubory typu:“)
    - Příklad: `"Text files (*.txt)|*.txt|All files (*.*)|*.*"`
  - `string Title` (Get, Set) – titulek dialogu (`Otevřít`)
  - `string InitialDirectory` (Get, Set) – Adresář po zobrazení dialogu
- Lze vybrat více souborů najednou – Vlastnosti:
  - `bool Multiselect` (Get, Set) – povolení výběru více souborů
  - `string[] FileNames` (Get, Set) – cesty + jména souborů, `Length` = počet vybraných souborů
- Reakce na výběr neexistujícího souboru (uživatel zadá jméno ručně), kontrola:
  - ručně přes `System.IO.File.Exists()`
  - automaticky (zobrazí varování) – `bool CheckFileExists` (defaultně `true`)
- Příklad:

```
OpenFileDialog opf = new OpenFileDialog();
opf.Filter = "Textové soubory (*.txt)|*.txt|Všechny soubory (*.*)|*.*";
opf.InitialDirectory =
      Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
      // cesta k adresari "Dokumenty"
DialogResult dr = opf.ShowDialog();
if (dr == DialogResult.OK)
{
```

```
            // uzivatel stiskl OK, nebo Enter
            // vyzvedneme si jmeno + cestu k souboru

            string soubor = opf.FileName;
        }
```

Grafika v .NET

- = vytváření grafických obrazců na formulářích
- Filosofie kreslení ve Windows



- GDI = Graphics Driver Interface
  - = API pro kreslení
- Kreslení je nezávislé na výstupním zařízení:
  - Obrazovka
  - Paměť
  - Tiskárna
  - …
- GDI+ = vylepšené GDI pro Windows XP a novější
  - Přístupné v .NET přes třídy ve jmenných prostorech:

```
System.Drawing
System.Drawing.Drawing2D
System.Drawing.Imaging
System.Drawing.Text
System.Drawing.Printing
```

- Chci kreslit = nutno požádat Windows o Device Context (DC, kontext zařízení)
  - Windows vytvoří dle možností zařízení (rozměry, rozlišení, barevná hloubka, možnosti ovladače…)
  - DC v .NET – reprezentován objektem třídy `System.Drawing.Graphics` = virtuální plátno
- Objekt třídy `Graphics` nelze libovolně vytvořit – nutno jej získat, možnosti:
  - Voláním `Control.CreateGraphics()` – přímé kreslení
  - Obsluhou události `Control.Paint`
- Princip 2D kreslení v .NET:
  - Grafická primitiva: `Color`, `Pen` (štětec), `Brush` (výplň), `Color` (barvy), `Rectangle` (obdélník – pozice a tvar), …
  - Nástroje pro kreslení: `DrawLine()`, `DrawRectangle()`, `DrawString()`, `FillRectangle()`…
    - Používají grafická primitiva

Přímé kreslení

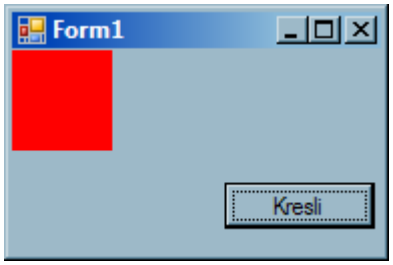
- Postup:
  - Získání DC
  - Kreslení
  - Navrácení zpět DC Windows (nutné!!!)
- Příklad:

```
private void buttonKresli_Click(object sender, EventArgs e)
{
	Graphics g = this.CreateGraphics();
	// 1) Graphics získán od Form - this
	try
	{
		Brush b = new SolidBrush(Color.Red);
		g.FillRectangle(b, 0, 0, 50, 50);	// 2) pouziti
	}
	finally
	{
		g.Dispose();				// 3) vraceni, nutne
	}
}
```

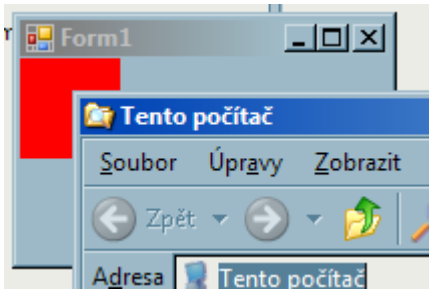
Lepší:

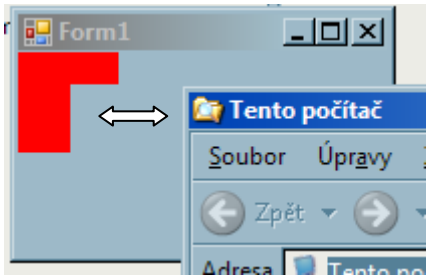
```
private void buttonKresli_Click(object sender, EventArgs e)
{
	using (Graphics g = this.CreateGraphics())
	{
		Brush b = new SolidBrush(Color.Red);
		g.FillRectangle(b, 0, 0, 50, 50);
		Dispose() zajistí „}" příkazu „using"
	}
}
```

- Výsledek:



- Ale pozor:





Činnost Windows při kresleni

- Skrytí (časti) okna – nutno jej opět obnovit:
  - Ovládací prvky, Titulkový pruh, … – obnoví Windows
  - Grafické objekty z `Graphics` **musí obnovit programátor**
- Událost `Paint` – zapouzdřuje zprávu WM_PAINT – zaslána aplikaci při požadavku Windows na obnovu vzhledu aplikace

Kreslení v události Paint

- `Graphics` je již vytvořen

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
        Graphics g = e.Graphics;
        Brush b = new SolidBrush(Color.Red);
        g.FillRectangle(b, 0, 0, 50, 50);
}
```

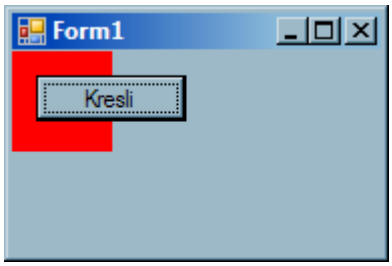
- Žádné `Dispose()` – `Graphics` nebyl vytvořen námi !!!
- Problém – vykresleno ihned po startu, nikoli na stisk tlačítka. Řešení:

```
public partial class Form1 : Form
{
        public Form1()
        {
                InitializeComponent();
        }

        bool vykreslit = false;

        private void buttonKresli_Click(object sender, EventArgs e)
        {
                vykreslit = true;
                this.Invalidate(true);    // reakce - Win zaslou WM_Paint
                // true - budou překresleny i podřízené prvky
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
                if (vykreslit)
                {
                        Graphics g = e.Graphics;
                        Brush b = new SolidBrush(Color.Red);
                        g.FillRectangle(b, 0, 0, 50, 50);
                }
        }
}
```

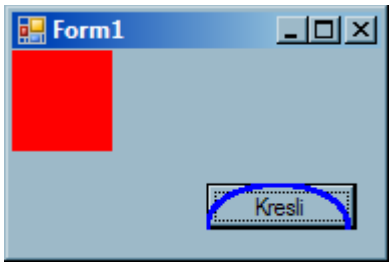
- Obsluhujeme `Paint` vždy toho prvku, na který chceme kreslit!!!
  - Kreslení na tlačítko = `Paint` tlačítka, nikoli `Form1`

Cíle kreslení

- Objekt `Graphics` dovolí kreslit jen na **Client Area** prvku, od kterého byl získán



- Kreslit lze na cokoli – `Graphics` pochází z `Control` (= kořen hierarchie prvků GUI)




```
private void buttonKresli_Paint(object sender, PaintEventArgs e)
{
	Graphics g = e.Graphics;  // g patří buttonKresli!!!
	Pen p = new Pen(Color.Blue, 3);
	g.DrawEllipse(p, 0, 0, 70, 40);
}
```

Činnost Windows při kresleni II

- Problém při změně velikosti okna

  - Windows překreslí jen to, co musí
  - Řešení:

```
public Form1()
{
	InitializeComponent();
	this.SetStyle(ControlStyles.ResizeRedraw, true);
	// Windows překreslí při změně velikosti vše
}
```

- Problém blikání obrazu:
  - Velký počet objektů k vykreslování – obraz bliká
  - Řešení = tzv. Double bufferring
    - Vykreslení obrazu nejprve do paměti
    - Celý obraz najednou na obrazovku

```
public Form1()
{
	InitializeComponent();
	// double buffering
	this.DoubleBuffered = true;
	this.SetStyle(ControlStyles.AllPaintingInWmPaint, true);
```

```
            this.SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
            this.SetStyle(ControlStyles.UserPaint, true);
        }
```