

# Vývoj aplikací v prostředí .NET

Katedra řídicí techniky,  
ČVUT-FEL Praha

3. přednáška

2.3.2011

K 13135, ČVUT FEL Praha

1

*Co počítač občas pilně dělá,  
když vlastně nic nedělá?  
Aneb, jak vznikají některé zprávy...*



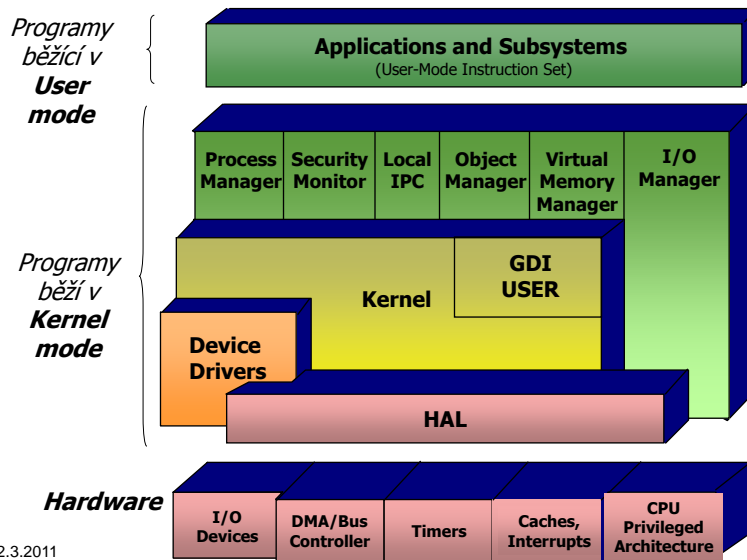
*Prerušujeme tento program kvůli rodičovské domluvě...*

2.3.2011

K 13135, ČVUT FEL Praha

2

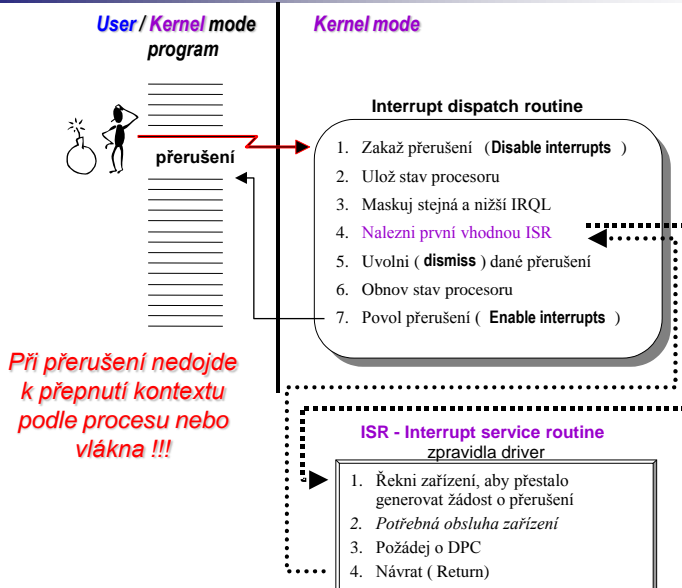
# Struktura operačního systému WinNT



2.3.2011

3

## Přerušení

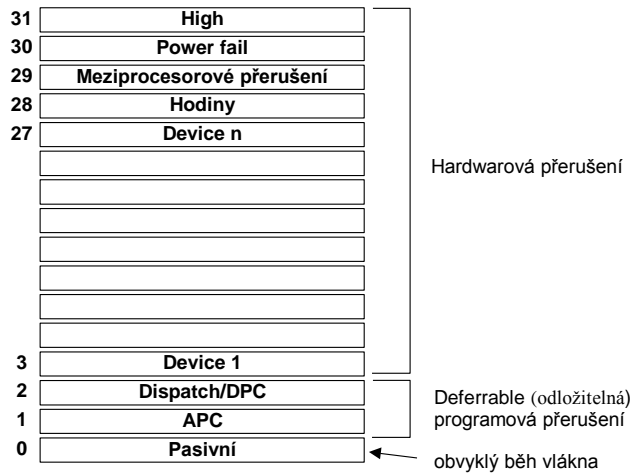


2.3.2011

K 13135, ČVUT FEL Praha

4

## Priorita přerušení skrze IRQL



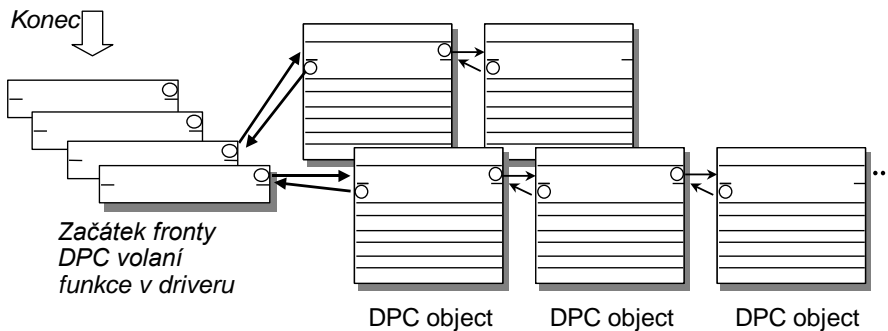
2.3.2011

K 13135, ČVUT FEL Praha

5

## DPC – Deferred Procedure Calls

- Odložené zpracování z vyšší IRQL na nižší "dispatch" úroveň
  - Driver zařadí požadavek do fronty
  - Pro každou CPU existuje jedna fronta
  - DPC se vykonávají po vyřízení vyšších IRQL



2.3.2011

K 13135, ČVUT FEL Praha

6

## Co to vlastně počítač provádí?

Administrative tools -> Performance Monitor : Add Counter

- **% DPC time** - čas spotřebovaný na obsluhu všech IRQL 2
- **% interrupt time** - čas spotřebovaný na obsluhu všech IRQL > 2
- **Interrupts/sec.** - počet přijatých přerušení za vteřinu
- **DPC queued/sec.** - počet odložených volání

*Pokud neběží žádný proces a systém není idle => přerušení či DPC*



[[http://www.uibk.ac.at/geotechnik/res/lame\\_II.html](http://www.uibk.ac.at/geotechnik/res/lame_II.html)]

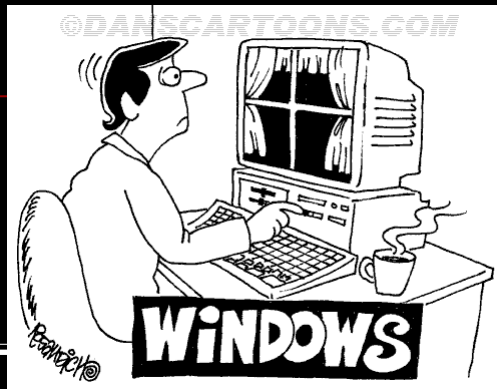
2.3.2011

K 13135, ČVUT FEL Praha

7

## Zprávy a události

*základ Windows a X-Windows*



2.3.2011

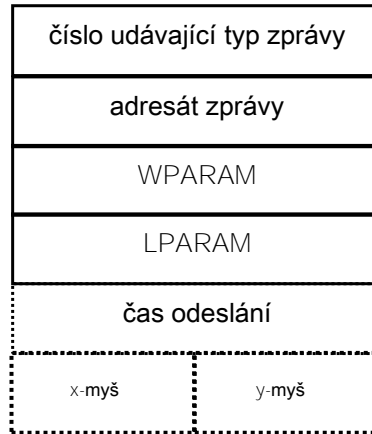
K 13135, ČVUT FEL Praha

8

## Co jsou to zprávy?

- zprávy = informace o událostech**  
navrženy 1984 v Massachusetts Institute of Technology pro X-Windows jako elegantní a jednoduché řešení pro menší množství sdílených dat.

6 \* Integer32



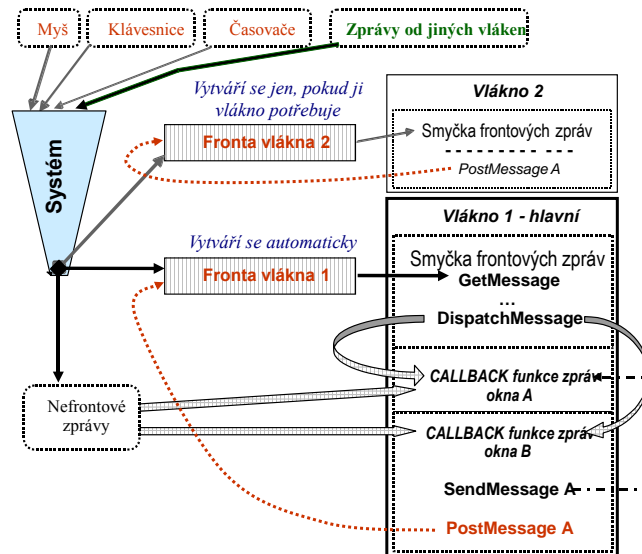
2.3.2011

K 13135, ČVUT FEL Praha

9

## Jak se zpracovávají zprávy?

Zjednodušené orientační schéma  
– přesnější popis procesu viz další snímky

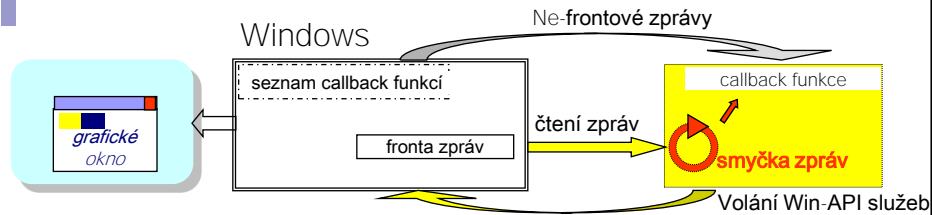


2.3.2011

K 13135, ČVUT FEL Praha

10

## Aplikace typu GUI - Graphical User Interface



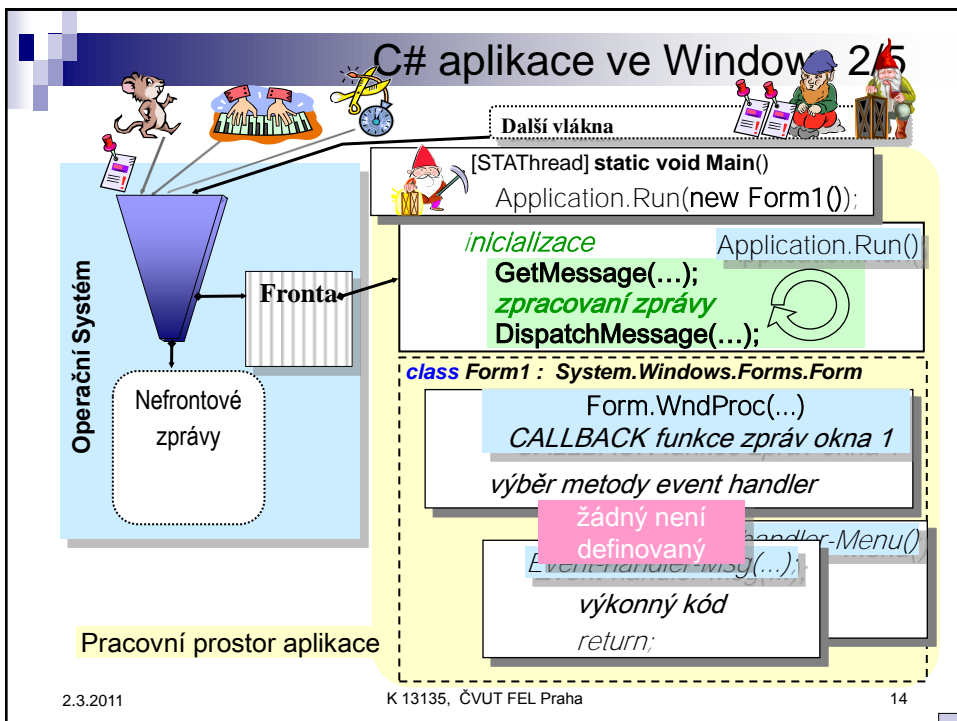
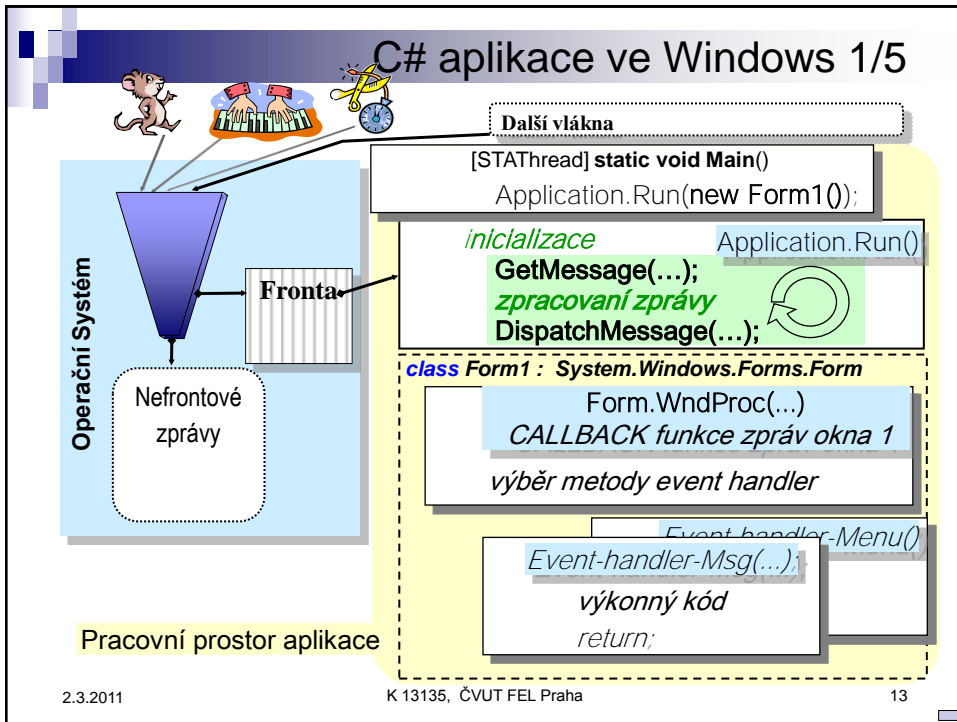
## Aplikace typu Console

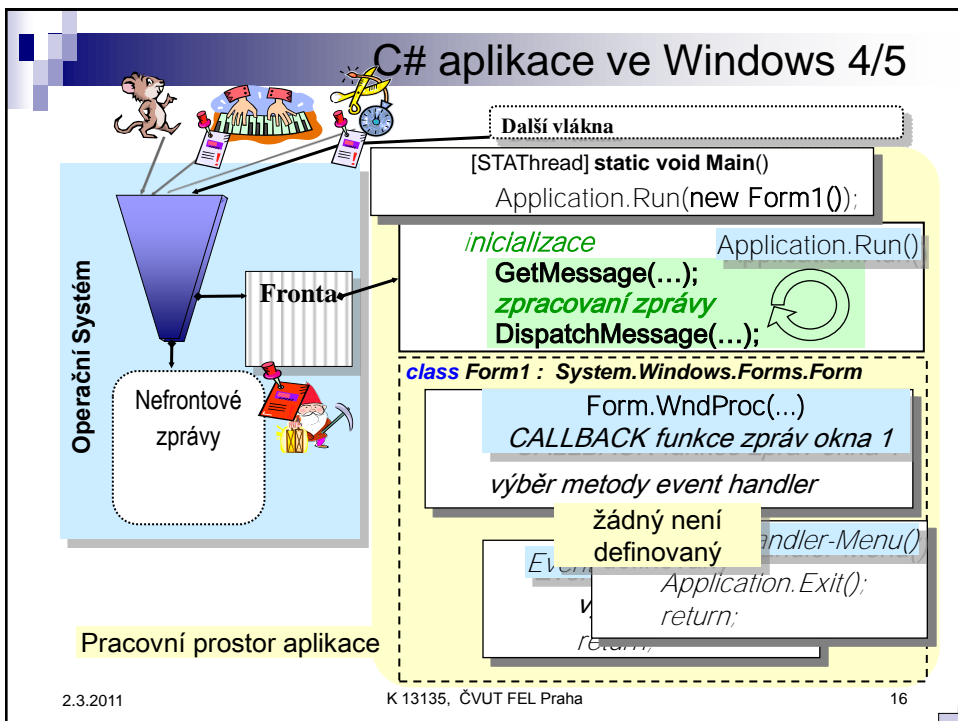
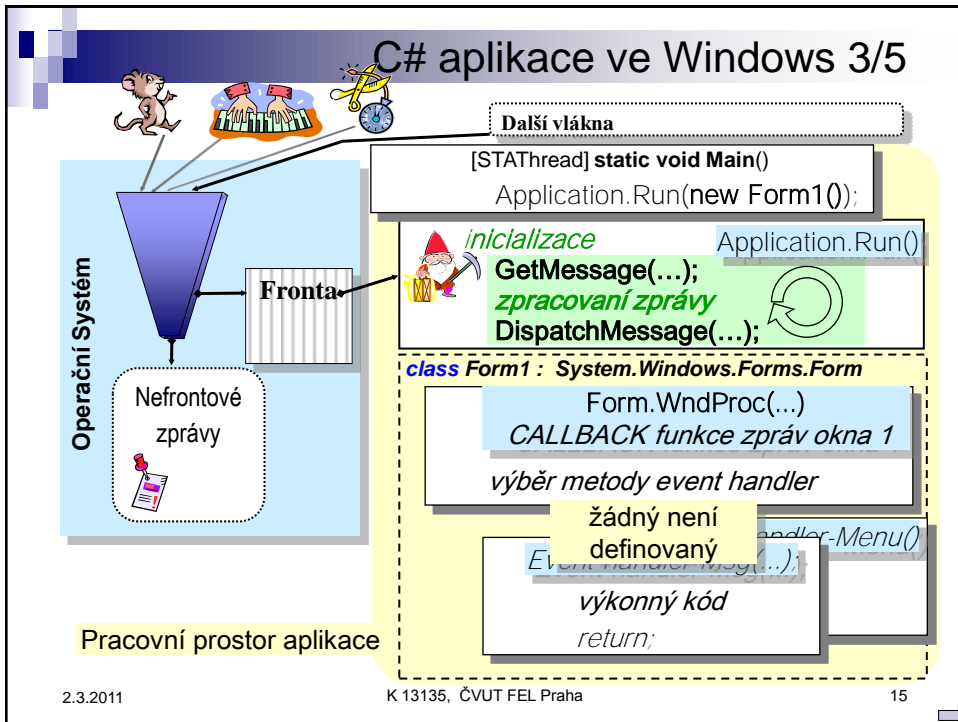


OS zpracovává smyčku zpráv, emuluje textové okno spolu se vstupním a výstupním proudem (pomocí Win API služby pipe)

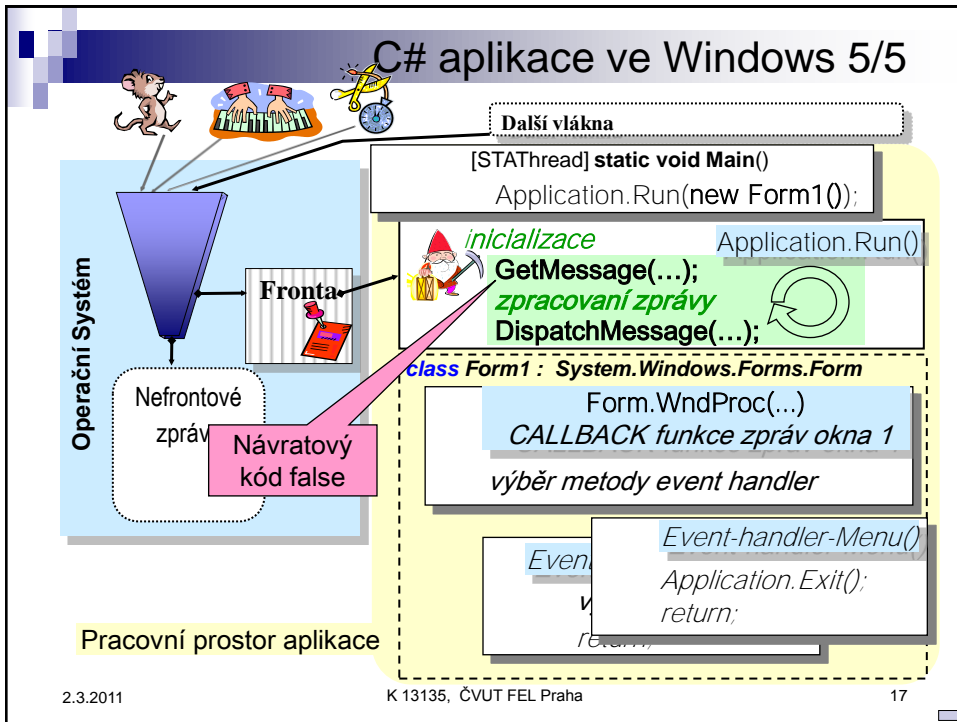
## Handler a Handle

- **Překlad pojmu Event Handler je v české literatuře značně nejednotný.**
  - **Použijeme raději nesklonný technický termín event handler, resp. ve zkratce pouze jako handler, pokud nebude hrozit záměna.**
- **Pozor, nezaměňujeme**
  - **handler** - ovladač realizovaný příkazy programu, zpravidla kódem metody
  - **handle** - číselný identifikátor přidělený OS vytvořenému prvku









## Některé systémové třídy

### System namespace

- **Application**  
 .Exit(), . ExecutablePath
- **Clipboard**  
 .GetDataObject() .SetDataObject()
- **Cursor**  
 .Current
- **Cursors**  
 .Arrow .Wait
- **Screen**  
 .GetWorkingArea()...
- **Environment**  
 .CommandLine .OperatingSystem .TickCount

2.3.2011 K 13135, ČVUT FEL Praha 18

## Shrnutí pro samostudium

- **Architektura OS Windows se opírá o strukturu zvanou server-klient.**  
V roli serveru vystupuje OS, zatímco jeho klienty jsou běžící procesy a jejich vlákna. Komunikace probíhá oběma směry. OS Windows vykonává operace zadané klienty přes služby Win API, dále zprostředkovává distribuci zpráv mezi klienty a některými jimi vytvořenými prvky. Klientům zasílá zprávy hlásící výskyt události.
- **OS předává většinu zpráv voláním zaregistrované callback funkce.** Pouze **frontové zprávy** (*queued messages*) se řadí do fronty (*paměti typu FIFO*), v níž čekají na zpracování podle pořadí jejich příchodu. Proces si musí sám čist frontové zprávy pomocí smyčky zpráv.
  - ta mu dovoluje i **vytřídit zprávy vyžadující speciální zpracování**, např. tzv. „Hot/Shortcut keys“ = funkční klávesy.
  - K frontovým zprávám se řadí všechny vstupní zprávy (např. od myši, klávesnice), dále zpráva WM\_PAINT (požadavek na překreslení okna), WM\_QUIT (ukončení aplikace), WM\_TIMER (hlášení o uplynutí intervalu časovače) a některé další.*
- Pozn. **TIMER má i nefrontovou verzi, tj. se zprávou přes callback funkci**

## Property



*Prodej bažinaté  
nemovitosti (property)  
o velikosti 1 akr*

[Z [www.grinningplanet.com/](http://www.grinningplanet.com/)]

## Java versus C#

### Java

```
private int radius;
public int getRadius()
{ return radius; }
public void setRadius(int value)
{ if (value < 0)
    radius = 0;
  else
    radius = value;
}

int s = kruh.getRadius();
kruh.setRadius(s+1);
```

### C#

```
private int radius;
public int Radius
{ get { return radius; }
  set {
    if (value < 0)
      radius = 0;
    else
      radius = value;
  }}

kruh.Radius++;
```

2.3.2011

K 13135, ČVUT FEL Praha

21

## Property - náhražka get/set metod

```
class Data {
  FileStream s;
  public string FileName {
    set { s = new FileStream(value, FileMode.Create); }
    get { return s.Name; }
  }
}

Data d = new Data();
d.FileName = "myFile.txt"; // → set("myFile.txt")
string s = d.FileName;    // → get()
```

**uložená hodnota** (pointing to `s`)

**typ property** (pointing to `string`)

**jméno property** (pointing to `FileName`)

**klíčové slovo: hodnota přiřazená do property** (pointing to `set`)

*JIT překladače často přeloží jako inline get/set metody  
→ není ztráta výkonu*

2.3.2011

K 13135, ČVUT FEL Praha

22



# Pro samostudium

## Následující snímky

- jsou určeny jako další rozšíření přednášky;
- projdeme je jenom zběžně
- a proto se nebudou ani zkoušet.



2.3.2011 K 13135, ČVUT FEL Praha 23

## Property jako náhrada datového členu

*Pro samostudium*

```
class C
{
    private static int size;
    public static int Size
    {
        get { return size; }
        set { size = value; }
    }
}
```

```
C.Size = 3;
C.Size += 2; // Size = Size + 2;
```

Můžeme si zde dát breakpoint na manipulaci s daty

2.3.2011 K 13135, ČVUT FEL Praha 24

## Vynechání get nebo set

*Pro samostudium*

```
class Account
{
    long balance;
    public long Balance { // pouze ke čtení
        get { return balance; }
    }
}
```

```
x = account.Balance;    // ok
account.Balance = ...;  // chyba
```

2.3.2011

K 13135, ČVUT FEL Praha

25

## C# Automatic Property >= .NET 3.0

*Pro samostudium*

```
public class .ane.nfo
{
    public string .ane { get; set; }
}
```

- Vytvoří se automaticky datový člen
- Význam pro interface

```
// test code
.ane.nfo g = new .ane.nfo();

// all set accessor
g.ane = "adiant ilvergun";

// all get accessor
System.Console.WriteLine(g.ane);
```

2.3.2011

K 13135, ČVUT FEL Praha

26

## Indexery = speciální typ properties

*Pro samostudium*

```
public class Clovek
```

```
{ private string jmeno; // datové členy properties private
```

```
  public Clovek(string text) { jmeno = text; }
```

```
  public string Jmeno { get {return jmeno; } }
```

```
  /* ... */
```

```
}
```

*Pozn.: Pokud by se 'jmeno' nastavovalo jen v konstruktoru, dal by se použít read only element*

```
public class Seznam
```

```
{ private Clovek[ ] osoby = new Clovek[4];
```

```
  public Clovek this [int index]
```

```
  { get { return osoby[index]; } }
```

```
  set { osoby[index] = value; } }
```

```
}
```

2.3.2011

K 13135, ČVUT FEL Praha

27

## Přístup na indexery

*Pro samostudium*

```
static void Main()
```

```
{
```

```
  Seznam lide = new Seznam();
```

```
  lide[0] = new Clovek("Pepa");
```

```
  lide[1] = new Clovek("Honza");
```

```
  lide[2] = new Clovek("Karel");
```

```
  lide[3] = new Clovek("Lucie");
```

```
  for (int i=0; i<4; i++)
```

```
    Console.WriteLine(lide[i].Jmeno);
```

```
}
```

2.3.2011

K 13135, ČVUT FEL Praha

28

- *používejte property, když se jedná logickou náhradu datového členu*

```
private string name;
```

```
public string Name
```

```
{ get { return name; }
```

```
  set { name = value.Trim(); }
```

```
        // úprava hodnoty před zápisem
```

```
}
```

- pro náročné operace - tím zdůrazníme vhodnost uchování jejich výsledku.
- pro konverze jako třeba **Object.ToString()**.
- když **get** člen má vedlejší efekt.
- když volání téhož členu dvakrát za sebou dává odlišné výsledky.
- když pořadí operací je důležité pro výsledek, u property se get a set mohou volat v libovolném pořadí při výpočtu výrazu.
- člen je statický, ale vrací hodnotu, která může být změněna.
- při vrácení pole - **property vracející pole jsou velice matoucí !**

## Vytvořte indexované property

*Pro samostudium*

- jen pro datové členy mající charakter pole.
- používejte pouze jeden index.
- jako index volte jediné typ integer nebo string.
- nevytvářejte indexované property a metody, které jsou semanticky ekvivalentní.
- pojmenujte Item ty elementy třídy, které jsou typu třída nebo struktura a mají definovaný indexer.

```
class Skupina
{ public read only string Jmeno;
  public class Seznam Item;
  /* */
}
```

2.3.2011

K 13135, ČVUT FEL Praha

31

## Delegate



*Rozšířený ukazatel na funkci,  
známý z jazyka C++*

2.3.2011

K 13135, ČVUT FEL Praha

32



## Hlavní prvky tříd

```
class Class1 {  
    ... fields, constants...           // pole, konstanty  
    ... methods...                     // metody  
    ... constructors...                // konstruktory,  
    ... destructors...                 // destruktory  
    ... properties ...                 // vlastnosti  
    ... indexers ...                   // indexery  
    ... delegates...                   // cca ukazatelé na metody  
    ... events ...                     // události  
    ... overloaded operators ...       // přetížené operátory  
    ... enums...                       // výčtové typy  
    ... class, struct...               // vnořené deklarace  
}
```

2.3.2011

K 13135, ČVUT FEL Praha

33

## C++ pointer na funkci

### 1. Metoda pro obsluhu události

```
double Mocnina(double d) { return d*d; }
```

### 2. Deklarace typu pointer na funkci

```
typedef double ( *LPMocnina)(double);
```

### 3. Definice proměnné typu pointer na funkci

```
LPMocnina lpMocnina;
```

### Testujeme

```
lpMocnina = Mocnina; // přiřadíme
```

```
d = (*lpMocnina) (5); // voláme
```

2.3.2011

K 13135, ČVUT FEL Praha

34

## Stejný program pomocí C# delegate

### 1. Metoda pro obsluhu události

```
double Mocnina(double d) { return d*d; }
```

### 2. Deklarace proměnné typu delegate

```
delegate double LPMocnina(double d);
```

### 3. Definice proměnné typu delegate

```
LPMocnina lpMocnina;
```

**//Testujeme**

```
lpMocnina = Mocnina;
```

```
double d = lpMocnina(5);
```

2.3.2011

K 13135, ČVUT FEL Praha

35

## Vlastnosti proměnné delegate

- Proměnná typu delegate může být i null, **ale pak se nesmí volat**, jinak dojde k výjimce.
- Proměnné typu delegate jsou objekty, takže se smí předávat jako parametry a použít jako členy tříd.
- Pokud se volá několik metod a typ delegate vrací hodnotu, tou bude výsledek posledního volání, což platí i pro případný parametr out.
- Parametr předaný delegate se použije pro všechny metody.

2.3.2011

K 13135, ČVUT FEL Praha

36

## Delegáty lze propojit třídy

```
class Prekladac // deklarční prostor Funkce
{
    private StringBuilder error = new StringBuilder();
    int errorCounter=0;
    private void VypisChybu(string text)
    { errorCounter++; error.AppendLine(text); }
    public delegate void Chyba(string text);
    public void Analyza(string code)
    { Parser p = new Parser(VypisChybu); p.Zpracuj(code); }
}

class Parser // deklarční prostor Parser
{
    Prekladac.Chyba vypisChyby;
    public void Zpracuj(string code) { if(code==null) vypisChyby("No code.");
                                     /* další operace */
    }
    public Parser(Prekladac.Chyba vypisChyby) {this.vypisChyby = vypisChyby;}
}
```

privátní metoda druhé třídy předaná jako klíč k přístupu

2.3.2011

K 13135, ČVUT FEL Praha

37

## Delegát jako předpis operace

```
class Program
{
    delegate void DoWithFile(string filename);
    static void ProcessDir(string directory,
                          DoWithFile doWithFile)
    {
        foreach (string file in Directory.GetFiles(directory))
            doWithFile(file);
        foreach (string dir in Directory.GetDirectories(directory))
            ProcessDir(dir, doWithFile);
    }
}

/* pokračování na dalším snímku */
```

2.3.2011

K 13135, ČVUT FEL Praha

38

## Prostý tisk

```
static void Main(string[] args)
{
    ProcessDir(@"C:\Temp", Print);
}
```

```
static void Print(string file)
{ Console.WriteLine(file); }
```

## Anonymní metoda

```
static void Main(string[] args)
{
    ProcessDir(@"C:\Temp",
        delegate(string file){ Console.WriteLine(file); }
    );
}
```

## Anonymous Methods

```
formN1.FormClosed      formální parametry  
+= delegate(type1 name1, type2 name2...)  
    { /*.....*/ };  
    kód metody
```

- zapíšeme přímo kód metody, ušetříme tvorbu pojmenované metody
- anonymní metoda má přístup na privátní členy svého objektu, podobně jako každý delegate
- return v anonymní metodě, znamená jen ukončení anonymní metody

### Omezení

- anonymní metoda nesmí být přiřazená objektu
- anonymní metoda nesmí přistupovat na ref a out parametry metody, která ji vytváří

2.3.2011

K 13135, ČVUT FEL Praha

41

## Možnost vynechat formální parametry

```
static void Main(string[] args)  
{  
    int count=0;  
    ProcessDir(@"C:\Temp",  
        delegate(string file){ count++; } );  
    count=0;  
    ProcessDir(@"C:\Temp", delegate{ count++; } );  
}
```

### Omezení

- formální parametry lze vynechat jen tehdy, pokud mezi nimi není nějaký parametr s modifikátorem **out**

2.3.2011

K 13135, ČVUT FEL Praha

42

## Delegate - možný seznam operací

```
static void Main(string[] args)
{
    DoWithFile operace = new DoWithFile(Print);
    int count = 0;
    operace += delegate { count++; };
    ProcessDir(@"C:\Temp", operace);
}
static void Print(string file) { Console.WriteLine(file); }
```

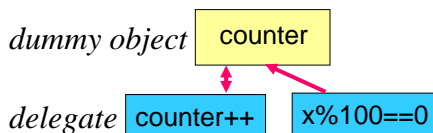
2.3.2011

K 13135, ČVUT FEL Praha

43

## Dummy proměnné

```
static void Main(string[] args)
{ ProcessDir(@"C:\Temp", CreateDemo()); }
static DoWithFile CreateDemo()
{ int counter = 0;
  DoWithFile operace
    = new DoWithFile( delegate(string file)
                      { Console.WriteLine(file); counter++; });
  operace += delegate {
    if (counter % 100 == 0) Console.WriteLine("/**100**/");
  };
  return operace;
}
```

*dummy object* 

*dummy* – *n.* atrapa, maketa, *adj.* fiktivní

*Proměnná counter je sdílena všemi delegáty, kteří ji používají, a existuje do konce životnosti posledního z nich.*

2.3.2011

K 13135, ČVUT FEL Praha

44

## Events - události

- Speciální případy typu delegate. *Jejich metody musí ale vracet void.*
- Pokud se používají pro obsluhu událostí ve Windows, mají navíc povinné dva argumenty
  - **object sender** – odesílatel zprávy
  - **EventArgs** třídu nebo třídu od ní odvozenou, např. **EventArgs**, **MouseEventArgs**, **PaintEventArgs**
- Deklarace event vytváří člen třídy určený pro uložení odkazu odkaz na event-handler

```
public delegate void EventHandler (  
    Object sender, EventArgs e );  
  
public event EventHandler MouseMove;  
  
Jméno členu typu event
```

2.3.2011

K 13135, ČVUT FEL Praha

45

## Jaký je rozdíl mezi event a delegate?

- **delegate** dovoluje operace **+=** **a** **-=**
- **event** má jen operace **+=** **a** **-=**
- **event** je systémově blízký property, pro tu překladač interně vygeneruje private datový člen typu delegate a skryje přitom operaci **=**
- **event** se může stát součástí deklarace interface (ta bude později), datový člen delegate nikoliv
- **public delegate** se může volat odkudkoliv.
- **public event** lze evokovat jen z jeho deklarčního prostoru třídy/struktury.

2.3.2011

K 13135, ČVUT FEL Praha

46

## Použití event

- **event** je cílený na GUI nebo jiné grafické aplikace, ač není apriori omezen jen tam.
  - Deklarací event dáváme najevo svůj úmysl – **obsluha události**.
- **event** omezuje manipulaci na přidání a ubrání metod, a proto z kódu odvozené třídy nelze zrušit již přidané private metody základní třídy, nutné pro správnou obsluhu událostí.
  - **Event chrání funkčnost základní třídy**.

*Příklady na event uvedeme hlavně  
v následující přednášce o oknech*

# Příští přednáška

## NETwork of ASP.NET

