Ošetřování chyb v programech

Úvod

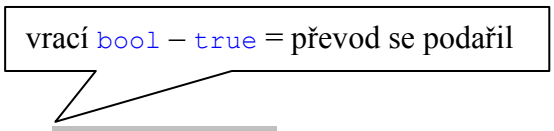
- chyba v programu = normální záležitost
- typy chyb:
  1) programátorská
     - chyba při návrhu
     - každých 10 000 řádek → 1 chyba
     - lze jen omezeně ošetřit (před pádem aplikace nabídnout uložení souborů, zaslání diagnostických informací)
  2) nevhodný zásah uživatele
     - např. program vyžaduje zadání čísla – uživatel zadá text
     - nutno plně ošetřit
- běhová chyba (run time error)

Způsoby ošetření chyb

- dány možnostmi programovacího jazyka
- Varianty:
  - Chybové (návratové) kódy
  - Výjimky

Chybové (návratové) kódy

- návratová hodnota metody = informace o běhu metody (typicky `bool`)




```
int cislo;
bool parseOK = Int32.TryParse(Console.ReadLine(), out cislo);
if (!parseOK)
      Console.WriteLine("prevod se nepodaril");
```

- Kontrola je dobrovolná, programátora k ní nic nenutí

```
int cislo;
Int32.TryParse(Console.ReadLine(), out cislo);
```

Výjimky (Exceptions)

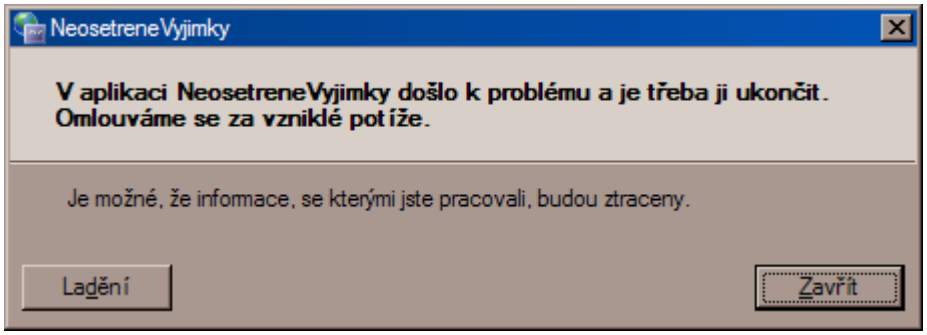
- = moderní způsob ošetření chyb v OOP jazycích
- Princip a základní pojmy:
  1) V programu dojde k běhové chybě
  2) Běh program se přeruší → **vyhodí se výjimka (throw exception)**
  3) Program se pokusí **výjimku zachytit** (**catch exception**) = ošetřit chybu ve spec. části kódu
     - kód neexistuje → program „spadne“ a skončí

- Příklad vyhození výjimky – načítání celého čísla z klávesnice (bude zadáno číslo reálné)

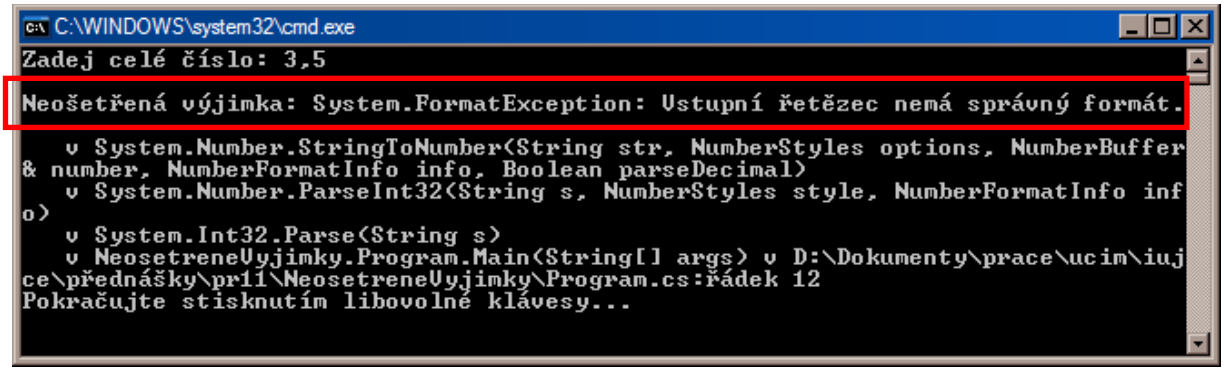
nebude zadáno celé číslo → vyhození výjimky

```
int a = Int32.Parse(Console.ReadLine());
```

1)



2)



Ošetření vyjímek

- příkaz `try` – `catch`

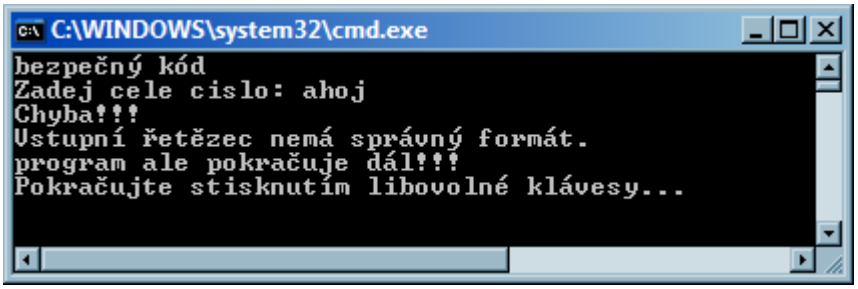
```
static void Main(string[] args)
{
    int x;
    Console.WriteLine("bezpečný kód");
    try
    {
        int a = Int32.Parse(Console.ReadLine());
        // Int32.Parse() několik typu vyjimek
        x = 255 / a;      // možná výjimka typu
                          DivideByZeroException
        Console.WriteLine("Vysledek deleni je {0}", x);

    }
    catch (Exception e)
    {
        Console.WriteLine("Chyba!!!");
        Console.WriteLine(e.Message);
    }
    Console.WriteLine("program ale pokračuje dál!!!");
}
```

výjimka = skok na ...

chráněný blok

zachycení výjimky

`string Message` – popis chyby



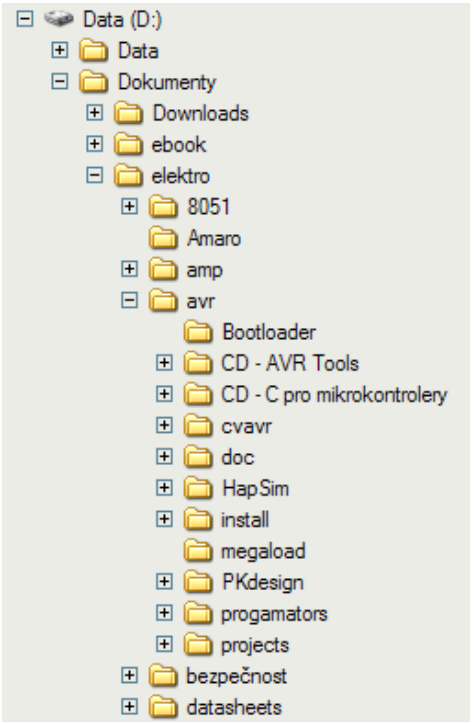
… a další

Způsoby reakce na chybu

- Jen vypíšeme hlášení (a ukončíme program → stejně se něco nepovedlo)
  - Viz. min. příklad
  - Jen v případech s „mizivou“ pravděpodobností výskytu
- Po zachycení problém napravíme (program pokračuje)
  - cyklus „dokud se bude vyhazovat výjimka, opakuj“

```
double prom;
while (true)
{
        try
        {
                prom = Double.Parse(Console.ReadLine());
                break;      // při chybě se to sem nedostane
        }
        catch (Exception e)
        {
                Console.WriteLine("Opakujte zadání, protože:");
                Console.WriteLine(e.Message);
        }
}
// vše OK, pokračujeme
```

Souborový vstup a výstup



Terminologie

- file = soubor;
- **adresář** (directory) – nebo také „složka“ (folder)
- **jméno souboru** (filename) = identifikace souboru na paměťovém médiu, skládá se z:
  - **přípona souboru** = vše od poslední tečky ve jménu do konce – .doc; .pdf
    - nemusí být
  - „jméno souboru“ = vše, co není přípona
  - délka jména:
    - MS-DOS: 8+3
    - Win32: max. 255 znaků
- **aktuální adresář** (current directory)
- **nadřazený adresář** (parrent directory)
- **kořenový adresář** (root)
- **relativní cesta** k souboru (relative path)
- **absolutní cesta** k souboru (absolute path)
- **úplné jméno souboru** = včetně cesty

Soubor

- soubor = data (posloupnost Bytů) uložená na nějakém paměťovém médiu
- práci se souborem na „nízké“ úrovni zajišťuje operační systém → programátor se nemusí o nic starat (např. fyzické uložení souboru apod.)
- typ souboru – „co to je za soubor?“
  - = přesný význam každého bytu v souboru pro jeho uživatele
  - rozlišení dle přípony souboru = název za poslední tečkou
- základní dělení na **textové a binární soubory**
- práce soubory je velmi podobná práci s obrazovkou a klávesnicí

Textové vs. binární soubory

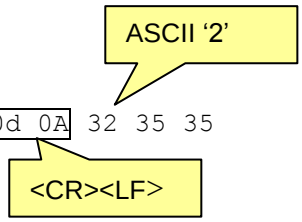
Textové soubory

- lze je zpracovat libovolným textovým editorem
  - mají čitelný obsah
  - nezaměňovat např. s textovým dokumentem ve Wordu – .doc není prostý text!!!
- přípony:
  - typicky `*.txt`
  - další ryze textové soubory: `*.html *.cs`
- jsou organizovány po řádcích

- různá délka
- končí domluveným znakem:
  - Windows → <CR><LF> (`"\r\n"`)
  - Linux → <LF> (`"\n"`)
  - Macintosh → <CR> (`"\r"`)

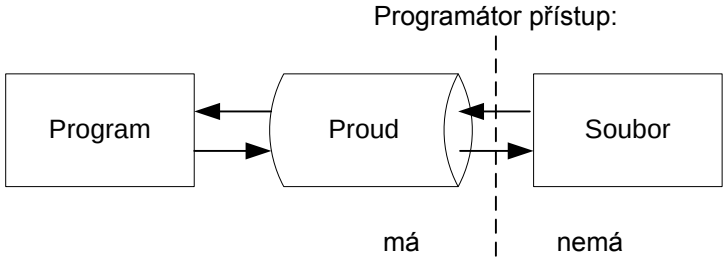
Binární soubory

- při běžné práci s PC mnohem častější
  - obrázky: .bmp .gif .jpg
  - zvuk: .mp3 .wav
  - video: .avi .mpeg .mov
  - dokumenty: .doc .xls .pdf
- vnitřní organizace („význam“ jednotlivých bytů) souboru závisí na tvůrci → **formát souboru**
  - veřejná specifikace formátu – video (.avi, .mpg), zvuk(.mp3, .waw), OpenOffice (.rtf)
  - neveřejná (.doc .xls, AutoCAD)
- oproti textovým: rychlejší zpracování, paměťově úspornější
- příklad obsahu souborů – uložení dvou čísel typu `char`: 1 a 255:
  - textový soubor (každé číslo na samostatném řádku): `31 0d 0A 32 35 35`
  - binární soubor: `1 FF`



Proudy (stream)

- proud = objekt k přenosu dat
  - přenos dat zvenčí do programu = čtení z proudu
  - přenos dat z programu ven = zápis z proudu

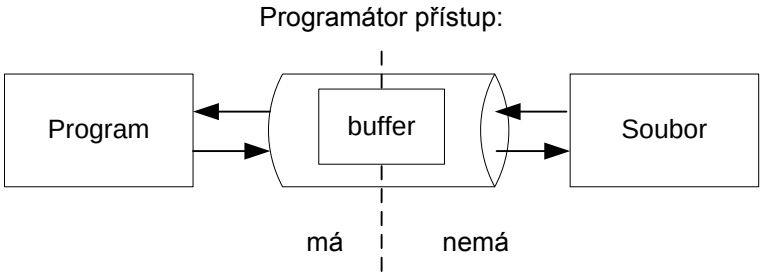


- princip práce nejenom se soubory – např. také síť, paměť, obrazovka (proud jen pro zápis), klávesnice (jen pro čtení)…
- výhoda: stejná logika pro různá zařízení → práce se souborem = práce s konzolí
  - kód může být nezávislý na zdroji/příjemci dat – např. lze přesměrovat vstup/výstup z konzole na soubor

Proudy s vyrovnávací pamětí (buffered stream)

- např. zápis souboru na disk po znacích → neefektivní

- lepší: zápis do RAM, a pak najednou větší blok na disk (podobně čtení)



- problém – v programu: zápis do souboru (všechno co jsem chtěl) → pád programu → vše se nemuselo zapsat na disk (něco v bufferu) → nutnost korektně práci se souborem ukončit
  - viz. např. vyjmutí flash disku z PC

Práce se souborem v C#

- manipulace na úrovni OS (přesuny, mazání …)
- práce s obsahem (čtení, zápis …)
  - základní principy stejné pro oba typy souborů:
    1) otevření souboru
    2) práce se souborem
       - čtení, zápis
       - pohyb po souboru – seek
    3) uzavření souboru
- třídy `StreamReader` a `StreamWriter`
  - jmenný prostor `System.IO` → nutno `using System.IO;`
- vždy možný výskyt běhové chyby → veškeré operace se souborem do `try`
- čeština bez problémů

Práce se souborem na úrovni OS

- Třída `File` – statické metody
- ```
  void File.Copy(string co, string kam, bool prepsat)
  void File.Copy(string soubor, string kam)
  ```
  - kopie souboru `soubor` na místo `kam`
  - `prepsat` – pokud `false` a soubor `kam` existuje → výjimka `IOException`
  - např. `File.Copy(@"c:\dokumenty\pr10.doc", "pr11.doc");`

    do aktuálního adresáře
- ```
  void File.Move(string soubor, string kam)
  ```
  - přesun souboru `soubor` na místo `kam`
- ```
  void File.Delete(string soubor)
  ```
  - vymaže soubor
- ```
  bool File.Exists(string soubor)
  ```

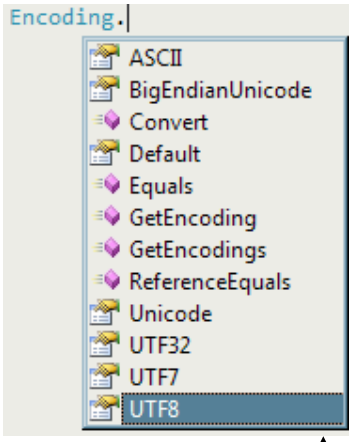
  - test existence souboru
- další metody viz. MSDN (atributy souboru, časy modifikace …)
- Třída `Directory` = práce s adresáři (jinak stejné jako `File`)
- Existují také `FileInfo` a `DirectoryInfo`

Kódování

- Detekce Unicode vs. 8b sady bez problémů (dáno principy)
- unicode volitelně na začátku souboru značka (BOM – byte order mask)
  - Textový soubor s obsahem a

kódování	Obsah (byty hex)
ASCII	61
1250	61
UTF-8	EF BB BF 61
UTF-16 LE	FF FE 61 00
UTF-16 BE	FE FF 00 61

- 8b sady nutno vybrat ručně (např. ASCII vs. 1250)
- Poznámkový blok:
  - Vytváří 1250
  - Čte (= detekuje) 1250, všechny Unicode



Čtení z textového souboru

Otevření a uzavření souboru

- Otevření = vytvoření proměnné souboru
- ručně nastavené kódování

```
StreamReader sr = new StreamReader(string jmeno, Encoding kodovani);
```

1) `jmeno` = (cesta +) název souboru (např. `@"C:\text.txt"`)
2) `kodovani` = použité kódování znaků
   - pro windows 1250 – `Encoding.GetEncoding(1250)`
   - ostatní – vlastnosti třídy `Encoding` (např. `Encoding.ASCII`)

- automatická detekce kódování (pro Unicode soubory)

```
StreamReader sr = new StreamReader(string jmeno, bool autodetekce);
```

1) `autodetekce` → nastavit `true`
2) 8b sady nelze rozeznat → nastaveno UTF-8

- Soubor neexistuje → výjimka (nutno ošetřit)

Uzavření souboru

- Uzavření – po skončení práce (viz buffered streams)
- způsoby:
  - instanční metoda `Close()`

```
try
{
        StreamReader sr = new StreamReader("aaa.txt", true);
        // prace se souborem
        sr.Close();
}
catch (Exception e)
{
        Console.WriteLine(e.Message);
}
```

  - příkaz `using`
    - soubor je uzavřen automaticky
    - preferovaný způsob

```
try
{
        using (StreamReader sr = new StreamReader("aaa.txt", true))
        {
                // prace se souborem
        }       // okamzik uzavreni souboru
}
catch (Exception e)
{
        Console.WriteLine(e.Message);
}
```

Možnosti čtení

- nelze náhodný přístup (ruční pohyb po souboru) – pouze sekvenčně od začátku do konce
- vše instanční metody (dále předpokládáme proměnnou `sr` typu `StreamReader`)
- `string ReadLine()` → přečte jeden řádek a posune ukazatel pozice v souboru na další
  - konec souboru → metoda vrací `null`
  - příklad: čtení souboru po řádcích a výpis na obrazovku

```
string radek;
while ((radek = sr.ReadLine()) != null)
{
        Console.WriteLine(radek);
}
```

- `int Read()` → přečte jede znak a posune ukazatel pozice v souboru na další
  - obvyklé použití `char znak = (char)soubor.Read()`
- `string ReadToEnd()` → přečte soubor od akt. pozice do jeho konce
  - příklad: čtení souboru a výpis na obrazovku

```
string obsahSouboru = soubor.ReadToEnd();
Console.WriteLine(obsahSouboru);
```

Zápis do textového souboru

Otevření souboru

```
StreamWriter sw = new StreamWriter(string jmeno, bool rezim, Encoding kodovani);
```

- kde:
  - `bool rezim` = způsob otevření:
    - `false` a soubor existuje → soubor je přepsán
    - `true` a soubor existuje → přidávání na konec
    - jinak vytvořen nový soubor
  - ostatní viz. čtení

Možnosti zápisu

- `sw.WriteLine()`, `sw.Write()` – Principy, formátování atd. viz `Console`