Výjimky

Jana Finkeová

Obsah

Výjimky

- přes všechny snahy o zabránění chybám za běhu programu může nastat neočekávaná situace, která způsobí pád programu
- některé chyby, které mohou nastat, nemůže programátor ovlivnit
  - přístup k souboru, který neexistuje
  - přístup k prvku pole mimo jeho rozsah
  - může být požadováno více paměti, než je možné uvolnit
  - chyba při aritmetických operacích
    - přetečení
    - dělení nulou
- **zachycení (ošetření) výjimek** (**Exception handling**) představuje mocný mechanismus pro zotavení programu z (většinou) neočekávaných stavů

Vyvolání výjimky

- snahou je **oddělit** od sebe kód, který představuje hlavní tok programu, od kódu pro zpracování chyb
- v okamžiku vzniku výjimečného stavu je vyvolána výjimka pomocí objektu, který obsahuje informace o daném stavu, aby bylo možné nalézt zdrojový problém
- tento objekt je instancí **třídy výjimky**, která může být uživatelsky definovaná, nebo lze využít třídy výjimek z knihovny tříd (FCL - Framework Class Library, někdy se také užívá zkratka BCL - Base Class Library).
- **tříd výjimek existuje velké množství**

Některé třídy výjimek

- **System.SystemException** - třída výjimek, které vyvolává rámec .NET, od této třídy je odvozeno mnoho tříd výjimek
- **System.ArgumentException** - odpovídající výjimka je vyvolána, jestliže alespoň jeden argument metody neodpovídá předepsaným specifikacím
- **System.ArgumentNullException** - odpovídající výjimka je vyvolána, jestliže alespoň jeden argument metody představuje nulovou referenci, přičemž by měl obsahovat referenci na existující instanci

Některé třídy výjimek

- **System.ArithmeticException** - odpovídající výjimka je vyvolána, jestliže dojde k neplatné aritmetické operaci či přetypování
- **System.DivideByZeroException** - odpovídající výjimka je vyvolána, jestliže dojde k dělení nulou
- **System.IndexOutOfRangeException**
- **System.FormatException**
- **System.OverflowException** - odpovídající výjimka je vyvolána, jestliže dojde k přetečení aritmetického typu v kontextu checked

Blok **try**, **catch** a **finally**

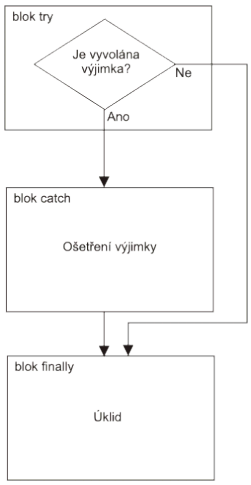
Doporučení:

1. kód pište do bloku **try**,
   pokud některý z příkazů nevyvolá výjimku, provedou se po spuštění všechny příkazy bloku,
   pokud dojde k nějaké chybě, přejde program do příslušného bloku **catch**
2. obslužné rutiny **catch** pro zpracování všech možných chyb pište bezprostředně za blok **try**; obslužných rutin může být víc pro ošetření konkrétních chyb
3. blok **finally** zajistí, aby program běžel i po výskytu výjimky, tento blok se vykoná vždy

Syntaxe

```
try
 {
   // kód programu , který může vyvolat výjimku
 }
 catch()
 {
   // zachycení (ošetření) výjimky
 }
 finally
 {
   // úklid
 }
```

Schéma



Příklad 1 – K jaké chybě může dojít?

```
try
 {...
   string precteno;
   int hodnota = Int32.Parse(precteno);
   …
 }
 catch (FormatException fEx)
 {
   WriteLine(fEx.Message);
 }
```

- může dojít k chybě při převodu řetězce na číslo
- jestliže řetězec obsahuje neplatné znaky, vyvolá metoda **Int32.Parse** výjimku **FormatException**
- proměnná **fEx** bude naplněna podrobnými informacemi o výjimce, položka **Message** obsahuje textový popis chyby

Příklad 1- K jaké další chybě může dojít?

```
try
 {...
   string precteno;
   double hodnota = Covert.ToDouble(precteno);
   hodnota = hodnota * hodnota;
 }
 catch (FormatException fEx)
 {
   WriteLine(fEx.Message);
 }
 catch (OverflowException oEx)
 {
   WriteLine(oEx.Message);
 }
```

- číslo může ležet mimo rozsah hodnot povolených pro čísla
- pak se vyvolá výjimka **OverflowException**

Obecná výjimka

- pro všechny možné výjimky, které mohou nastat existuje obecný typ výjimky **Exception**
- můžete tedy za blok **try** umístit více obslužných rutin **catch**, specifické pro konkrétní výjimky a obecnou pro všechny ostatní
- vyvolané výjimce pak může odpovídat více obslužných rutin umístěných za blokem **try**
- pak je důležité pořadí, spustí se první obslužná rutina, která odpovídá danému typu výjimky
- proto je potřeba nejprve psát obslužné rutiny pro konkrétní výjimky a teprve za ně umístit rutiny obecnější
  - pokud byste napsali rutinu pro FormatException až za Exception, ta by se nikdy nevyvolala

Příklad 1 výjimky několika typů

```
try
 {...

 }
 catch (FormatException fEx)
 {
   WriteLine(fEx.Message);
 }
 catch (OverflowException oEx)
 {
   WriteLine(oEx.Message);
 }
 catch (Exception Ex) // obecná obslužná rutina
 {
   WriteLine(Ex.Message);
 }
```


Příklad

- v cyklu požadujeme, aby uživatel zadal celé číslo, případně Enter pro ukončení programu
- vstup je reprezentován jako řetězec, struktura System.Int32 má metodu **Parse**, která má za úkol převést řetězec na celé číslo daného typu (**Convert** pro převod na různé datové typy)
- v případě, že uživatel zadal číselnou hodnotu, se daný převod zdaří, jinak vyvolá zmíněná metoda výjimku, kterou je potřeba zachytit:
  - Kód, ve kterém může být vyvolána výjimka, umístíme do bloku **try**.
  - Pokud je vyvolána výjimka, pokračuje program blokem **catch**.
  - Na konci lze umístit blok **finally**, který je vykonán buď po bloku try, pokud nedošlo k žádné výjimce, nebo po bloku catch, jestliže výjimka byla vyvolána. Blok finally často slouží k úklidu použitých prostředků, jako například k uzavření připojení k souboru.

Příklad

```
string vstup
    while (true)
      {
       try
         {
          Console.Write("Zadejte celé číslo (nebo Enter pro ukončení): ");
          vstup = Console.ReadLine();
          if (vstup == String.Empty) break;
               //  převést zadaný řetězec na celé číslo
          int cislo = int.Parse(vstup); // nebo: int cislo = Convert.ToInt32(vstup);
          Console.WriteLine("Zadali jste {0}.", cislo);
         }
        catch (Exception ex)
         {
           // V bloku catch zachytíme výjimku a vypíšeme text výjimky:
           Console.WriteLine("Zachycená výjimka: {0}", ex.Message);
         }
        finally
         {
             Console.WriteLine("Děkujeme za zadání.");
         }
       }
```


Přetečení celočíselných operací

- základní číselné datové typy mají danou maximální a minimální velikost
- např. **int** má rozsah hodnot -2 147 483 648 až 2 147 483 647
  - minimální či maximální hodnotu můžete zjisti pomocí **Int32.MaxValue** nebo **Int32.MinValue**
- kód neprovádí kontrolu a umožňuje tzv. tiché přetečení
- když potřebujete kontrolu zapnout, máte k dispozici klíčová slova **checked** a **unchecked**
- kód, který budete chtít kontrolovat, umístíte do bloku **checked**
- pokud dojde k přetečení, vyvolá se výjimka **OverflowException**
- Proč tato kontrola není standardní součástí jazyka?
  - většinu výpočtů není třeba kontrolovat
  - prodlužuje výpočet

Příklad

```
int cislo = Int32.MaxValue;
checked
        {
         int vyvolaVyjimku=cislo++;
          }
```


checked a **unchecked** ve výrazech

- obě klíčová slova lze použít i přímo v celočíselných výrazech
- klíčové slovo se předřadí před výraz a ten se uzavře do kulatých závorek

```
int vyvolaVyjimku= checked (Int32.MaxValue  + 1 );
int nevyvolaVyjimku= unchecked (Int32.MaxValue  + 1 );
```

Příklad - příkaz **throw**

- metoda NazevMesice, která podle zadaného čísla vrací název odpovídajícího měsíce jako řetězec (při zadání NazevMesice(1) vrátí textový řetězec Leden...)
- co má metoda vrátit, když bude zadaná hodnota menší než 1 či větší než 12
- ať vyvolá výjimku **ArgumentOutOfRangeException**
- ve větvi **default** je použit příkaz **throw**
- vytvoří se nový objekt výjimky a objekt **Message** se naplní zadaným textovým řetězcem

Příklad – příkaz **throw**

```
public static string NazevMesice(int mesic)
{
 switch (mesic)
  {
   case 1:
    return :”Leden”;
    ....
   case 4:
    return :”Duben”;
    .....
   default:
     throw new ArgumentOutOfRangeException(“Měsíc chybně”);
  }
}
```

Příště: Statické metody