Výjimky a validace

Úvod

V této kapitole si řekneme, jak v C# ošetřovat nesprávné vstupy od uživatele a jak pracovat s výjimkami. Pro tyto účely nám poslouží třída `Exception`.

K čemu slouží výjimka

Výjimky slouží k ošetření krajních situací při běhu programu a případnému ukončení programu alespoň s vypsáním chybové hlášky. Jejich používání by se tedy nemělo přehánět. Obecně platí, že pokud náš kód vyvolává fatální výjimky při běžném používání, je správně jej přepsat tak, aby i v krajních situacích fungoval alespoň na základní úrovni a nebylo nutné jej ukončit. Správné použití výjimek by mělo být spíše pro debuggovací (tzn. testovací) účely. Pokud program poskytneme uživatelům, měla by se výjimka objevit jen tehdy, když nastane situace, kterou jako programátoři nemůžeme ovlivnit. Mezi takovéto situace můžeme řadit například absenci administrátorských práv, absence důležitých balíčků .NET frameworku a tak dále.

 # Výjimka jako objekt

Jak jsme si již v předchozích kapitolách řekli, C# je silně objektově orientovaný programovací jazyk. Nemělo by tedy být překvapením, že výjimky jsou také objekty a je tedy možné s nimi podle toho pracovat - tedy volat jejich metody a pracovat s jejich vlastnostmi. Je také možné vytvářet vlastní výjimky, kromě používání již vestavěných. C# má však bohatou zásobu výjimek a pro naše účely by nemělo být nutné přidávat žádné vlastní. Každá výjimka je v jazyce C# potomkem třídy `Exception`. Přesný význam toho, co je to potomek třídy se dozvíte později v kapitole `dědičnost`. Prozatím si jenom řekneme, že výjimek existuje velké množství a všechny vychází z již zmíněné třídy `Exception`.

Jak vyvolat výjimku?

Každému už se to určitě alespoň jednou podařilo. Z edukativních důvodů si ale ukážeme například `DivideByZeroException`. Jak název napovídá, tuto chybu můžeme vyvolat při pokusu o dělení číslem nula. Vše si ukážeme na jednoduchém příkladě:

Dělení nulou

```
// Naše pomocná proměnná
int i = 0;

// Pokus o dělení nulou
int x = 14 / i;
```

Každý jistě z hodin matematiky zná, že v oboru reálných čísel nemá takováto operace žádný smysl. Nulou zkrátka nelze dělit. Stejného názoru je i kompilátor jazyka C# a pokud takováto situace nastane, dojde k neočekávanému ukončení programu a následnému vyhození výjimky `DivideByZeroException`. Jazyk C# nám ale dává možnosti, jak se s neočekávanými situacemi tohoto typu vypořádat. To může být užitečné v momentě, kdy například programujeme kalkulačku a nechceme, aby náš program při dělení nulou padal.

Blok try-catch

Blok `try-catch` nám umožňuje zachytit výjimku hned po jejím vzniku a na danou situaci patřičně reagovat.

Výjimky

```
try
{
   int i = 14 / int.Parse(Console.ReadLine());
}
catch
{
   Console.WriteLine("Nelze dělit nulou.");
}
```

V případě zadání čísla `0` by tedy program dále pokračoval vypsáním řetězce "Nelze dělit nulou." Za výraz catch můžeme ještě dopsat kulaté závorky s libovolným výjimkovým typem čímž docílíme, že odchytávat budeme pouze právě zmíněný typ:

Zachycení konkrétní výjimky

```
Form form = null;

try
{
   int x = 100 / form.Size.Width;
}
catch (DivideByZeroExpception ex)
```

```
{
   Console.WriteLine(ex.Message);
}
```

Takovýto fragment kódu by vyvolal `NullReferenceException`, neboť objekt (proměnná) `form` není inicializována - odkazuje na hodnotu `null`. Ačkoliv je celé tvrzení obaleno `try` blokem, dojde i tak k pádu programu a vyhození výjimky, neboť blokem `catch` odchytáváme pouze výjimky typu `DivideByZeroExpception`. V těle bloku `catch` pak vypisujeme obsah chybového sdělení za pomocí vlastnosti `Message`.

Vlastní vyvolání výjimky

Jazyk C# umožňuje vyvolat v krajních situacích výjimku a to buď naši vlastní nebo již předvytvořenou. Vše ilustruje následující ukázka:

Vyvolání výjimky

```
throw new Exception("Něco se nepovedlo...");
```

Klíčové slovo `throw` značí, že vyvoláváme výjimku. Následuje klíčové slovo `new`, jelikož vytváříme novou instanci třídy (objekt) a na závěr následuje název třídy, který označuje, jakou výjimku vyvoláváme. V tomto případě se jedná o obecnou generickou výjimku. Do konstruktoru výjimky můžeme předat řetězec popisující chybu.

Validace vstupů pomocí tryParse

Pojďme si nyní představit metodu `TryParse()`. Tato metoda slouží pro konverzi mezi datovými typy. Přebírá dva argumenty. Prvním je hodnota, kterou chceme přetypovat a druhým je proměnná, do které se má uložit výsledek konverze. Návratovým typem této metody je `bool`, který signalizuje, zda konverze proběhla v pořádku, či nikoli:

Bezpečné přetypování

```
// Hodnota, jenž budeme převádět na int
string hodnota = "23abc";
int vysledek;

// Přetypování
bool uspech = int.TryParse(hodnota, out vysledek);
```

Pojďme si nyní rozebrat tento fragment kódu. Na začátku inicializujeme proměnnou, která je typu `bool` a do které bude uloženo, zda konverze proběhla v pořádku. Samotnou metodu `TryParse()` můžeme volat z jakéhokoliv primitivního datového typu. Protože náš řetězec `23abc` se chceme

pokusit převést na datový typ `int`, voláme metodu právě z tohoto datového typu. Prvním argumentem je řetězec, který se budeme pokoušet převádět a druhým argumentem je proměnná, do které se v případě úspěšného převodu uloží přetypovaná hodnota. Povšimněte si, že druhému argumentu předchází klíčové slovo `out`. Metoda `TryParse()` nemůže přetypovanou hodnotu vracet, neboť již vrací `bool` symbolizující úspěšnost či neúspěšnost konverze. Potřebujeme-li tedy, aby metoda vracela více jak jednu hodnotu, klíčové slovo `out` je cesta, jak toho docílit.

V případě naší ukázky bude konverze neúspěšná, neboť náš řetězec obsahuje písmena. Proměnná `uspech` by tedy nabyla hodnoty typu `false` a proměnná `vysledek` by zůstala nezměněná.

Shrnutí

V této kapitole jsme si objasnili práci s výjimkami a třídou `Exception`. Jejich hierarchii, syntaxi, jak je vyvolávat a ošetřovat. Nakonec jsme si řekli, jak pracovat se uživatelským vstupem a ošetřit nesprávné vstupy od uživatele pomocí metody `TryParse()`.

Revision #2
Created 2025-05-21 08:53:53 UTC by Magdalena Dobešová
Updated 2025-05-21 10:22:13 UTC by Magdalena Dobešová