Metody

Úvod

V minulých kapitolách jsme se seznámili s proměnnými a se základními operacemi, které s nimi můžeme provádět. V této kapitole si představíme pojmy `třída` a `metoda`.

Metody

Metoda je úsek kódu, který můžeme opakovaně spouštět. Jazyk C# (přesněji framework .NET) v základu disponuje velkým množstvím metod, které Microsoft předpřipravil pro usnadnění naší práce. Jako první z nich si ukážeme metodu `Console.Beep()`, na které si budeme demonstrovat základy nové látky:

Metoda

```
Console.Beep();
```

Po spuštění (odborně `zavolání`) této metody se ozve krátký zvuk. Často se budeme setkávat s tím, že volaná metoda bude přijímat jeden nebo více tzv. `parametrů`. To je i případem naší metody:

Metoda s parametry (1)

```
// Pípnutí o frekvenci 450Hz a
// délce 100 milisekund
Console.Beep(450, 100);
```

Metoda s parametry (2)

```
// Proměnné
int frekvence = 450;
int delka = 100;

// Pípnutí o frekvenci 450Hz a
// délce 100 milisekund
Console.Beep(frekvence, delka);

// Argumenty ve spojení s operátory
```

```
Console.Beep(100 * 2, (100 % 60) / 2);
```

Metoda umožňuje zadat celkem dva parametry typu `int` – frekvenci a délku pípnutí. Konkrétní předaná čísla `450` a `100` jsou v tomto případě tzv. `argumenty` naší metody, které píšeme vždy do kulatých závorek. Pojmy parametr a argument jsou velmi často nesprávně zaměňovány. Pokud metoda přejímá více parametrů, jednotlivé argumenty pak vždy oddělujeme čárkou (`,`). Argumenty můžeme do metod pochopitelně předávat i formou proměnných (ukázka 2). Musíme pouze dohlédnout na to, aby nám korespondovali datové typy mezi předávanou proměnnou a mezi odpovídajícím parametrem metody.

> Při volání metod záleží na pořadí argumentů! – První argument odpovídá prvnímu parametru, druhý argument odpovídá druhému parametru a tak dále.

Některé parametry mohou být `nepovinné`. Nepovinné parametry nalezneme vždy až na samotném konci všech parametrů. Nestane se tedy, že by za nepovinným parametrem následoval parametr povinný.

> Pro úplnost ještě uvedu, že výchozí hodnoty parametrů metody `Console.Beep()` jsou 800 Hz a 200 ms.

Častěji se ale budeme setkávat s metodami, jejichž parametry budou povinné a jejich vynechání bude znamenat chybové hlášení. Narazit můžeme dokonce i na kombinaci obou výše zmíněných případů. Celá situace ohledně parametrů se ještě navíc komplikuje tím, že jedna metoda může mít různé kombinace parametrů, které je schopna pojmout. O takovýchto metodách říkáme, že jsou tzv. `přetížené` (anglicky overloaded methods). Dobrým příkladem může být naše výše zmíněná metoda `Console.Beep()`:

Přetížená metoda

```
// První overload
Console.Beep();


// Druhý overload
Console.Beep(450, 100);
```

Ještě doplním, že není možné zavolat metodu pouze s jedním argumentem nesoucí frekvenci nebo délků trvání. Bylo by to možné pouze za podmínky, že by existoval třetí overload metody, který by vyžadoval pouze jeden parametr. Při volání metod musíme tedy vždy respektovat jednotlivé variace metod (overloady) a pořadí jejich parametrů.

Setkat se můžeme i s `pojmenovanými argumenty`. Ty nám výrazně usnadní práci v případech, kdy si nepamatujeme přesné pořadí argumentů nebo chceme zvýšit přehlednost našeho kódu:

Pojmenované argumenty (1)

```
// Pojmenované argumenty
Console.Beep(frequency: 450, duration: 100);
```

Další výhodou tohoto zápisu je, že pořadí argumentů pak můžeme i prohazovat:

Pojmenované argumenty (2)

```
// V případě pojmenovaných argumentů nemusíme
// dodržovat přesné pořadí parametrů. Argumenty
// můžeme libovolně prohazovat.
Console.Beep(duration: 100, frequency: 450);
```

Pojmenované argumenty jsou validní pouze pokud nejsou následovány argumenty klasickými (`pozičními`) a od C# 7.2 pokud jsou poziční argumenty použity ve správném pořadí:

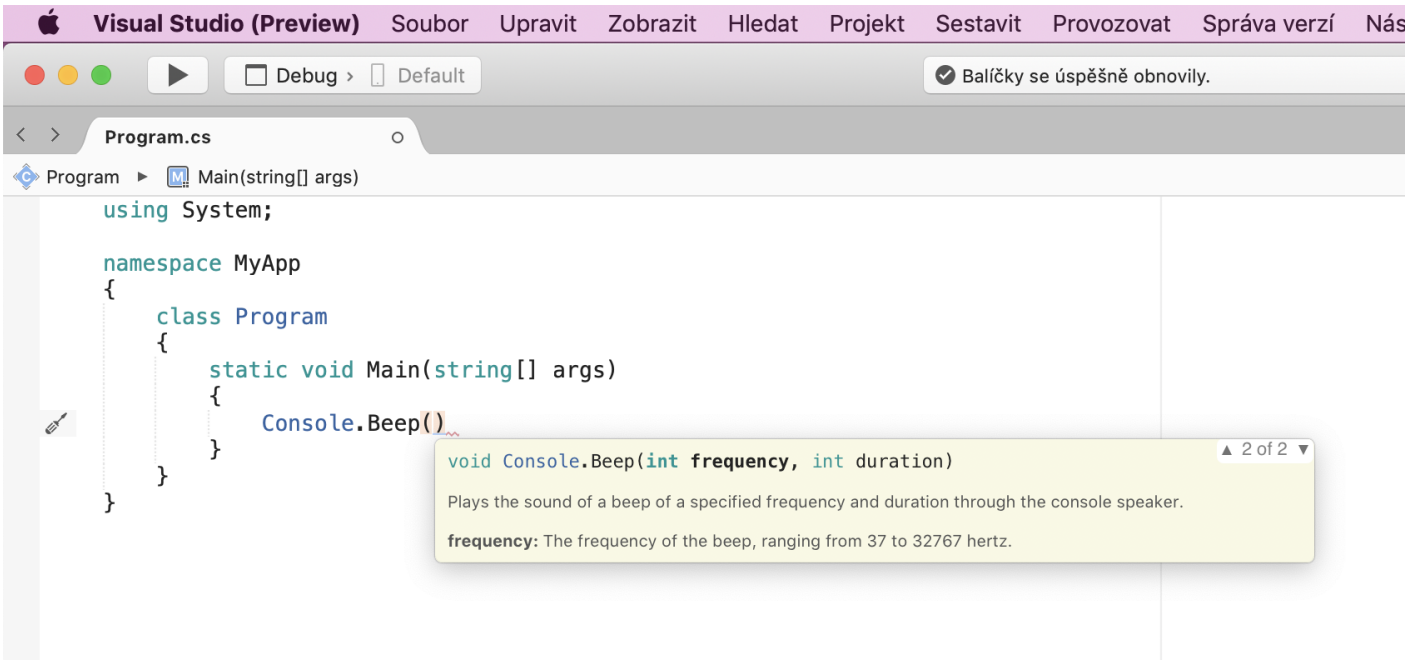
Pojmenované argumenty (2)

```
// Validní kód. Pojmenovaný argument není
// následován argumentem pozičním.
Console.Beep(450, duration: 100);

// Validní kód pro C# verze 7.2 a vyšší.
// Pojmenovaný argument je sice následován
// argumentem pozičním, ale byla zachována
// správná pozice vzhledem k pořadí
// parametrů v metodě.
Console.Beep(frequency: 450, 100);

// Chybný kód. Za pojmenovaným argumentem
// se nachází argument poziční. Nebyla
// dodržena ani správná pozice druhého
// argumentu - frekvenci píšeme na první
// pozici.
Console.Beep(duration: 100, 450);
```

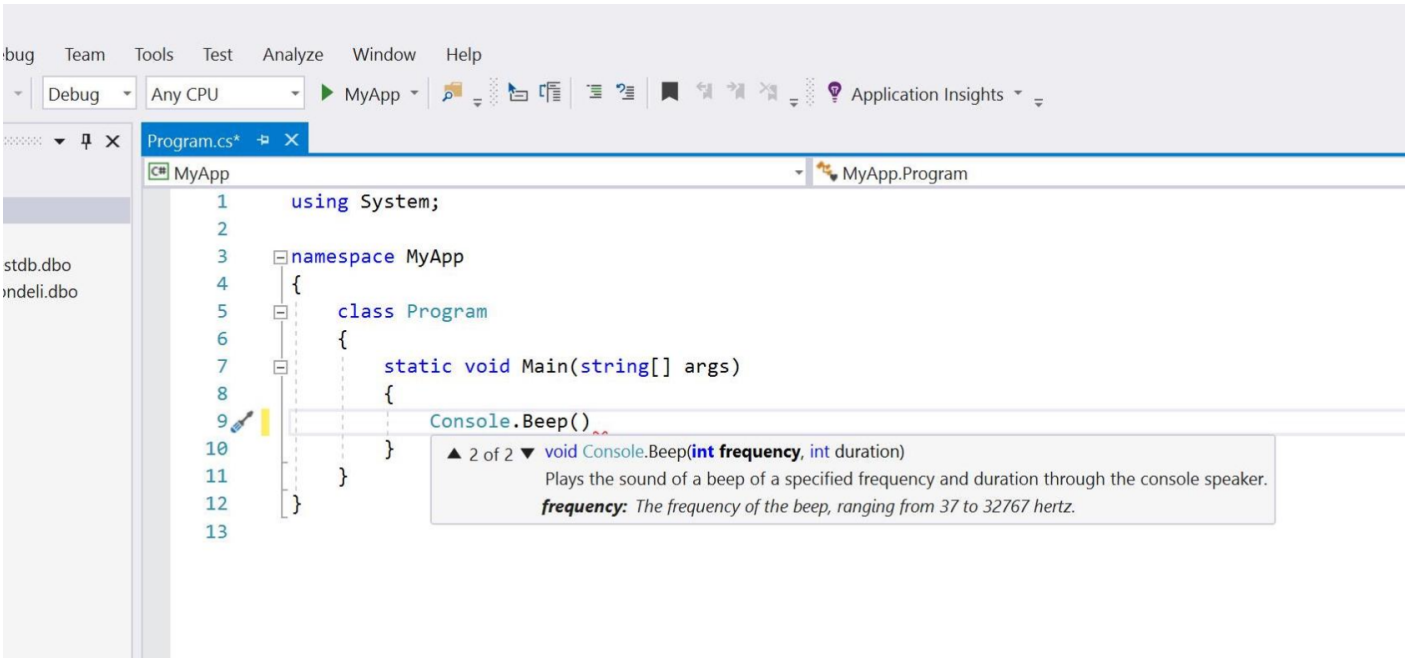
> Pojmenované argumenty jsou spíše rozšiřující látkou a běžně se s nimi setkávat nebudeme.

Jak jsme se nyní sami přesvědčili, situace kolem metod a jejich parametrů může být občas velmi komplikovaná. Úkolem programátora pochopitelně není znát zpaměti přesné pořadí a možné kombinace všech parametrů. Naši práci většinou výrazně zjednoduší vývojové prostředí, které nám všemožně napovídá a urychluje tedy proces celého vývoje. Souhrnu takovýchto chytrých funkcí říkáme obecně `intelligent code completion` (inteligentní doplňování kódu). Vývojové prostředí

Visual Studio má vlastní implementaci tohoto systému nazvanou `IntelliSense`, na kterou se nyní blíže podíváme. IntelliSense za nás chytře dokončuje slova nebo zobrazuje pomocné nabídky. Kdykoliv například dopíšeme kulaté závorky za konec metody, zobrazí se nám list se všemi možnými overloady dané metody a výčet jejich parametrů:



Ukázka IntelliSense nabídky ve Visual Studio for Mac.



Ukázka IntelliSense nabídky ve standardním Visual Studiu pro Windows.

Kromě stručného popisu metody nám vyskakovací list ještě napovídá jednotlivé parametry a jejich význam. Šipkami se pak můžeme pohybovat mezi jednotlivými overloady metody. Pokud bychom narazili na nepovinný parametr, bude za jeho datovým typem a názvem ještě operátor přiřazení (`=`) a jeho výchozí hodnota. Jedinou neznámou je pro nás slovo `void` na samém začátku listu. Jedná se o tzv. `návratový typ` metody. Tento konkrétní návratový typ značí, že po zavolání metody

nečekáme nazpět žádný výsledek. Metoda se prostě provede a pokračuje se na další řádek. Některé metody ale po svém zavolání mohou vrátit hodnotu, kterou můžeme poté odchytit a uložit do proměnné. Vše si ukážeme na příkladu:

Návratová hodnota metody

```
int x = Math.Abs(-27); // x = 27
```

Jak název napovídá, metoda `Math.Abs()` vrátí absolutní hodnotu předaného čísla a uloží jej do proměnné `x`. Návratová hodnota je pak v tomto případě typu `int`. Pro upřesnění ještě dodám, že metoda může vracet jakýkoliv datový typ. Záleží pouze na konkrétní metodě a její implementaci. Pokud metoda žádnou hodnotu nevrací, návratovým typem je pak vždy `void`.

> Později si ukážeme, jak napsat naší vlastní metodu.

Jako další si nyní ukážeme metodu `Console.WriteLine()`. Tato metoda přijímá parametr typu `string`, jehož obsah pak vypíše na nový řádek konzole. Pokud bychom zavolali bezparametrický overload této metody, vypíšeme jednoduše prázdný řádek a další budoucí výpis se bude provádět až po mezeře (prázdném řádku):

Nový řádek

```
// Bezparametrická verze metody - vypíše
// prázdný řádek
Console.WriteLine();

// Parametrická verze metody. Vypíše předaný
// řetězec
Console.WriteLine("Ahoj světe!");
```

Pokud bychom se nyní pokusili náš kód otestovat, okno konzole se zavře téměř okamžitě po spuštění. Důvodem je, že po vypsání našeho řetězce program končí a okno konzole se tedy zavře. Dodatečným voláním metody `Console.ReadKey()` zajistíme, že po vypsání řetězce bude program ještě čekat na stisknutí libovolné klávesy. Tím zabráníme samovolnému zavření okna. Náš program tedy nyní vypadá nějak takto:

Program

```
using System;
□□□□□
public class Program
{
□public static void Main()
□{
```

```
        // Vypíše řetězec do konzole
□□Console.WriteLine("Ahoj světe!");

□□// Počká na stisk libovolné klávesy
□□Console.ReadKey();
□}
}
```

Rovněž se někdy můžete setkat i s voláním metody `Console.ReadLine()`. Ta uživateli umožní napsat celý řetězec (na místo jednoho znaku) a samotné potvrzení je provedeno klávesou enter. Uživatelem zadaný řetězec je pak vrácen jako návratová hodnota typu `string` a může být poté uložena do proměnné. Pokud vrácenou hodnotu neuložíme, nic závažného se nestane a uživatelem zadaný řetězec je pak zahozen. Pro úplnost ještě zmíním existenci metody `Console.Write()`, která předaný řetězec píše na stejný řádek, jako předchozí výpis. Na malé ukázce si nyní budeme demonstrovat použití metody `Console.ReadLine()`:

Vstup od uživatele

```
// Získá textový řetězec od uživatele
string vstup = Console.ReadLine();
```

Takovýto kus kódu počká, až uživatel zadá textový řetězec a po stisknutí klávesy enter se uživatelem zadaný řetězec uloží do proměnné vstup a program poté dále pokračuje.

> Metoda `Console.ReadLine()` **VŽDY** vrací datový typ string a to i v případech, kdy uživatelem zadaný řetězec je pouze číslo (např. `82`). Jakákoliv úprava vstupu, validace (ověření správnosti) nebo přetypování je pak prací programátora.

Vlastní metody

V jazyce C# můžeme psát i naše vlastní metody. To se velmi hodí v situacích kdy víme, že daný úsek kódu budeme používat na více místech. Metody pro nás budou nástrojem, jak minimalizovat duplicity v kódu a dekomponovat (rozložit) logiku naší aplikace do více snadněji spravovatelných celků. Při zakládání nové metody kromě názvu vždy určujeme i její parametry a návratovou hodnotu. Vše si ukážeme na následujícím příkladu:

Deklarace proměnné

```
using System;


namespace MojeAplikace
{
```

```
    class Program
    {
        // Hlavní metoda Main. Je automaticky spuštěna
        // po startu programu.
        public static void Main(string[] args)
        {
            // Volání naší nové metody
            Pozdrav();
        }

        // Naše nová metoda
        public static void Pozdrav()
        {
            Console.WriteLine("Ahoj!");
        }
    }
}
```

Metody vždy píšeme na úroveň třídy. V našem případě budou metody vždy definovány ve třídě `Program`, neboť jsme si ještě neukázali, jak zakládat třídy jiné. Při definici metody začínáme tzv. modifikátorem přístupu `public`, který označí metodu jako veřejnou. Taková metoda je pak viditelná mimo její rodičovskou třídu (v našem případě třídu `Program`). Opačným modifikátorem je pak `private`, který naopak zařídí, že naše metoda mimo třídu nebude viditelná a půjde zavolat jen v té dané třídě, ve které se nachází. Modifikátory přístupu se nyní nebudeme blíže zabývat a naše metody vždy budeme označovat jako `public`, než si v pozdějších kapitolách tuto problematiku rozebreme více do podrobna. Obdobně budeme postupovat i u modifikátoru `static`, jenž bude předmětem poslední kapitoly. Oba modifikátory tedy budeme brát jako nutné „zlo“, než si v pravý čas osvětlíme jejich význam. Dále pak následuje návratová hodnota metody. Ta může být buď typu `void` (metoda nevrací žádný výsledek viz příklad výše) nebo může vracet libovolný datový typ (například `int` viz ukázka níže). Dále pak uvádíme název metody a kulaté závorky, jejichž význam si za chvíli osvětlíme. Metoda pak vždy končí složenými závorkami, jež definují její tělo a do kterých píšeme samotnou funkčnost naší metody. Nyní si ještě ukážeme metodu s parametry a návratovou hodnotou:

Deklarace proměnné

```
using System;

namespace MojeAplikace
{
    class Program
    {
        // Hlavní metoda Main. Je automaticky spuštěna
```

```
        // po startu programu.
        public static void Main(string[] args)
        {
            // Volání naší nové metody
            Pozdrav();
        }

        // Naše nová metoda
        public static void Pozdrav()
        {
            Console.WriteLine("Ahoj!");
        }

        // Metoda s parametry a návratovou hodnotou
        public static int Secti(int a, int b)
        {
            return a + b;
        }
    }
}
```

Na této ukázce jsme naše řešení ještě doplnili o metodu se jménem `Secti`, která jako dva své parametry očekává čísla typu `int` a za pomocí klíčového slova `return` vrací jejich součet. Klíčové slovo return ukončí činnost metody a vrátí hodnotu výrazu, který jsme za toto slovo dopsali (v našem případě součet proměnných `a` + `b`). V praxi to znamená, že jakýkoliv další kód po tomto klíčovém slově již nebude vykonán. Pokud jsme metodě definovali návratovou hodnotu jinou než `void` (tzn. očekáváme od metody nazpět nějaký výsledek), musíme v těle metody vždy klíčové slovo `return` použít. Samotné parametry metody pak vždy oddělujeme čárkou a nezapomínáme určit jejich datový typ. Parametry pak v těle metody vystupují jako obyčejné proměnné, s kterými můžeme libovolně manipulovat. Na závěr už jen doplním, že nezáleží na pořadí metod uvnitř třídy (`class`). Metody mohou být volány v jiných metodách nezávisle na svém pořadí.

> V případě všech datových typů, které jsme se zatím naučili (tzv. primitivních datových typů) platí pravidlo, že veškeré změny provedené na parametrech metody se neprojeví mimo metodu samotnou. Vše lépe osvětlí následující příklad:

Deklarace proměnné

```
using System;

namespace MojeAplikace
{
```

```
    class Program
    {
        public static void Main(string[] args)
        {
            // Naše pomocná proměnná
            int x = 0;

            // Volání metody, jež upravuje obsah proměnné
            Metoda(x);

            // Vypíšeme obsah proměnné
            Console.WriteLine(x); // => 0
        }

        public static void Metoda(int x)
        {
            // Změníme hodnotu předané proměnné
            x = 1;

            // Vypíšeme hodnotu proměnné
            Console.WriteLine(x) // => 1
        }
    }
}
```

Tento kód by po spuštění nejdříve vypsal `1` a pak `0`. Nejdříve by se tedy provedl výpis v těle metody `Metoda`, kde hodnota `x` je po změně z nuly rovna jedné a poté výpis v těle metody `Main`, kde hodnota proměnné `x` je stále `0`, neboť do těla metody je předána pouze samotná hodnota proměnné a ne odkaz (reference) na proměnnou samotnou. Tato problematika je předmětem primitivních a referenčních datových typů a budeme se ji zabývat podrobněji v poslední kapitole.

Setkat se můžeme i se situací, kdy metoda volá sama sebe. Jedná se o tzv. rekurzi a existují případy, kdy je toto chování velmi užitečné. Rekurze ale musí být vždy nějak ukončena, neboť by v opačném případě vznikl nekonečný cyklus, jež by vedl k přetečení zásobníku a pádu programu. V programování tuto chybu nejčastěji označujeme jako stack overflow a byla po ní pojmenována i velmi oblíbená a známá webová stránka pro vývojáře.

Třídy

Než prozatím uzavřeme kapitolu metod, uvedeme si ještě pár teoretických faktů a rozšíříme si naše dosavadní portfolio operátorů.

Pozornější čtenáři si mohli povšimnout, že samotný název metody (např. `Beep` nebo `WriteLine`) je kromě kulatých závorek doplněn i o tečku (`.`), které předchází vždy ještě nějaký název (v našem případě `Console` nebo třeba `Math`). Odborně hovoříme o tzv. `třídách` a naše tečka je pro nás novým operátorem sloužícím pro přístup ke členu. V třídách sdružujeme metody, které spolu logicky souvisí. Třída `Math` například slouží jako kontejner pro všechny metody souvysejícími s matematickými procedurami. Třída `Console` pak zaštiťuje všechny důležité nástroje pro manipulaci s konzolí. Například tedy výstup (výpis na konzoli) nebo získávání vstupu od uživatele. Operátor členu (`.`) pak slouží jako vstupní brána, která nám po svém napsání odhalí obsah dané třídy. Třída kromě metod může obsahovat i další členy, jako třeba nám dobře známé proměnné. Dále pak i vlastnosti nebo události, o kterých se budeme bavit v budoucích kapitolách. Setkat se můžeme dokonce i s třídou, která uvnitř sebe definuje další třídu (poté hovoříme o tzv. nested class, kterými se nebudeme v tomto materiále zabývat). Na těchto členech můžeme poté opět volat operátor členu (`.`) a prohlížet si prostřednictvím IntelliSense nabídek obsah daného členu ještě více do hloubky. Prozatím si ale vystačíme pouze se znalostí zavolat z dané třídy námi požadovanou metodu, než se blíže seznámíme z dalšími členy tříd.

Na závěr už jen zmíním, že v jazyce C# mají třídy a metody vždy velké počáteční písmeno a stále je dodržována camel case syntaxe – jednotlivá slova tedy oddělujeme velkým počátečním písmenem na místo mezery. Díky počátečnímu velkému písmenu pak můžeme snadno odlišit název třídy od názvu proměnné. Metodu pak vždy bezpečně poznáme podle kulatých závorek.

Shrnutí

Metody jsou jedním ze základních stavebních kamenů naší aplikace. Při práci s metodami musíme dát pozor na důsledné dodržování pořadí jednotlivých parametrů. Konkrétní hodnoty dosazované za tyto parametry pak nazýváme argumenty. Nezapomínejme na to, že metody rovněž mohou po svém zavolání vracet návratovou hodnotu, která může (ale nemusí) být odchycena a uložena do proměnné.

Revision #2
Created 2025-05-21 06:54:20 UTC by Magdalena Dobešová
Updated 2025-05-21 08:12:33 UTC by Magdalena Dobešová