.NET C# - Extension Methods

Petr Novák / novakpe@fel.cvut.cz / 25.02.2021

Obsah

1 Úvod

Co to jsou „Extension Methods“ (Metody Rozšíření)? Pokud je potřeba do nějaké již existující třídy přidat novou metodu, tak jsou například tyto možnosti. První možností je tvorba vlastní třídy, poděděné od původní a vlastní třída poté přidává požadovanou novou metodu. Avšak co když je původní třída tzv. uzavřená (seal) a nedá se od ní podědit žádná vlastní třída. V tomto případě je nutno použít druhou možnost a tou jsou „Extension Methods“ (Metody Rozšíření).
Jde v podstatě o způsob jak vytvořit metody ve vlastním programovém kódu a kompilátoru „jako by“ podstrčit, že tyto metody jsou součástí původní třídy. Ano metody jsou pouze „jako by“ součástí původní třídy, ale s několika velkými rozdíly:

- Nové metody nemají přístup k vnitřním (privátním) datům původní třídy.
- Nové metody nemohou překrýt metody původní třídy.

Poznámky:

- Proč jsou nějaké třídy uzavřeny (seal)? Toto může být z několika důvodů, například:
  - V budoucnu se předpokládá změna vnitřní struktury třídy, a proto není dobré povolit uživatelům přístup k jejímu vnitřnímu obsahu.
  - Třída je dočasná a není dobré na ní hloubkově zakládat další části své práce.
  - …

2 Jak vytvořit Extension Methods

Představme si často používaný případ, kdy požadujeme převod stupňů na radiány nebo naopak. Můžeme toto vyřešit vytvořením statických metod **RadToDeg** a **DegToRad** následujícím stylem:

```
// prevod 'Deg -> Rad'
public static double DegToRad(double degrees)
       { return degrees * (Math.PI / 180.0); }

// prevod 'Rad -> Deg'
public static double RadToDeg(double radians)
       { return radians * (180.0 / Math.PI); }
```

Jejich použití je velmi snadné, například:

```
double uhelRad = 123;                   // uhel v radianech
double uhelDeg = RadToDeg(uhelRad);     // uhel ve stupnich
```

Double

Avšak číslo **double** je tvořeno třídou a obsahuje například metodu **ToString()**. Proč by nemohlo obsahovat i tyto dvě metody:

- **RadToDeg()** – Převod radiánů na stupně.
- **DegToRad()** – Převod stupňů na radiány.

Samozřejmě lze tohoto celkem snadno dosáhnout a to pomocí následujícího kódu:

```
namespace TestExtensionMethods
{
    // trida obsahujici rozsireni pro 'double'
    public static class DoubleExtensionMethods
    {
        // prevod 'Deg -> Rad'
        public static double DegToRad(this double degrees)
            { return degrees * (Math.PI / 180.0); }

        // prevod 'Rad -> Deg'
        public static double RadToDeg(this double radians)
            { return radians * (180.0 / Math.PI); }
    }
}
```

Název **NameSpace** může být libovolný. Rovněž název třídy může být zcela libovolný, ale je vhodné ji aspoň částečně pojmenovat podle třídy, kterou rozšiřuje, tedy třídy pro kterou jsou rozšiřující metody určeny, například **DoubleExtensionMethods**.
Jak třída, tak i rozšiřující metody musí být vždy **static**. V tomto případě nejde o skutečné metody, které jsou součástí cílové třídy (tedy její instance), ale pouze metody chovající „se jako“ statická součást cílové třídy.
Velmi důležité je klíčové slovo **this** udávající jakou cílovou třídu uvedená metoda rozšiřuje, tedy do jaké cílové třídy tato metoda bude staticky patřit. Klíčové slovo **this** je před **double** a tudíž metoda rozšiřuje, neboli patří do třídy **double**.
Původní kód pro převod jednotek bude mít nyní poněkud jiný zápis:

```
double uhelRad = 123;                   // uhel v radianech
double uhelDeg = uhelRad.RadToDeg();    // uhel ve stupnich
```

Na první pohled se zdá jako by metoda **RadToDeg()** byla součástí třídy uchovávající číslo **double**. Skutečnost je však poněkud jiná. Toto je pouze přehlednější tvar pro uživatele, ale ve skutečnosti tento zápis kompilátor převede interně na původní formu zápisu (pomocí zcela externí statické metody).

String

Dalším příkladem může být tvorba řetězce složeného opakováním vstupního znaku nebo řetězce. Za tímto účelem lze do třídy **String** přidat metodu **Repeat(…)** přijímající číslo určující kolikrát se má vstupní / původní obsah zkopírovat do výstupu. Zápis rozšiřující metody **Repeat()** bude následující:

```
namespace TestExtensionMethods
{
    public static class StringExtensionMethods
    {
        // vstupni retezec, pocet opakovani znaku / retezce
        public static string Repeat(this string str, int times)
        {
```

```
            // 'StringBuilder' pro efektivni cinnost
            System.Text.StringBuilder sb = new System.Text.StringBuilder();
            // opakovani puvodniho znaku / retezce zadanym poctem
            for (int i = 0; i < times; i++) { sb.Append(str); }
            // vraceni noveho retezce
            return sb.ToString();
        }
    }
}
```

Klíčové slovo **this** je uvedeno před **String** a tudíž tato metoda rozšiřuje třídu **String**, tedy „jako by“ patřila do třídy **String**. Použití metody **Repeat()** je stejné jako v předchozím případě:

```
string strIn = "ABC";               // puvodni retezec
string strOut = strIn.Repeat(3);    // novy retezec „ABCABCABC“
```

MyClasses

Rozšiřující metody lze samozřejmě použít na jakékoli třídy, tedy i na vlastní. Zde je uveden příklad, který tak jak je v podstatě nemá velký význam, ale je celkem vhodný pro pochopení častého důvodu tvorby rozšiřujících metod. Existuje třída **Clovek** obsahující nějaké vlastnosti a je z nějakého důvodu uzavřena proti dědění (sealed). Velmi často je však potřeba získat celé jméno člověka a to sloučením nějakých vlastností veřejně dostupných ve třídě. Proč tedy nevytvořit další metodu **FullName()** poskytující přímo celé jméno člověka a interně využívající jiné potřebné vlastnosti původní třídy.

```
// polozky jednoho cloveka
public sealed class Clovek
{
       public int Id { get; set; }
       public string FirstName { get; set; }
       public string LastName { get; set; }
}

// rozsirujici metody pro 'Clovek'
static class DomainEntityExtensions
{
       // ziskani celeho jmena
       public static string FullName(this Clovek value)
         => $"{value.FirstName} {value.LastName}";
}
```

Využití rozšiřující metody **FullName()** je nyní následující:

```
// instakce tridy 'Clovek'
Clovek clovek = new Clovek() { FirstName = "AAA", LastName = "BBB" };
// ziskani celeho jmena 'AAA BBB'
string fullName = clovek.FullName();
```