C#
Extension Methods

70-483 – Skill 2.1

Grundlagen

- Mit Erweiterungsmethoden (extension methods) lassen sich vorhandene Datentypen erweitern, ohne dass ein neuer Typ davon abgeleitet werden muss.
- Erweiterungsmethoden werden in statischen Klassen als statische Methoden deklariert, können aber auf einer Objektinstanz aufgerufen werden.
- Erkennbar sind sie am Schlüsselwort `this`, das immer vor dem ersten Parameter steht.
- Dieser erste Parameter legt den Typ fest, der erweitert wird.
- Sie können alle `public`-Member des erweiterten Typs nutzen.

Beispiel

statische Klasse

statische Methode

```
static class StringExtensions {
    public static string SwitchCase(this string s) {
        StringBuilder sb = new StringBuilder();
        foreach (char c in s) {
            sb.Append(char.IsUpper(c)
                ? char.ToLower(c)
                : char.ToUpper(c));
        }
        return sb.ToString();
    }
}
```

Klasse string wird erweitert

Aufruf auf einer Instanz von string

```
class Program {
    static void Main(string[] args) {
        string s = "Hello World!";
        s = s.SwitchCase();
        Console.WriteLine(s);
        Console.WriteLine("AbCdEf".SwitchCase());
        Console.WriteLine(StringExtensions.SwitchCase("xyz"));
    }
}
```

Aufruf auf einem Literal

Oder als statische Methode

Extension Members

- Seit C# 14 / .NET 10 gibt es endlich eine weitere Möglichkeit für Erweiterungen von bestehenden Typen.
- Mit Extension Members können jetzt zusätzlich auch statische Methoden sowie statische und Instanz-Properties erzeugt werden.
- Dazu wurde ein `extension(<receiver>)`-Block eingeführt.
- Die Extension Members werden dann innerhalb des Blocks für den angegebenen Receiver geschrieben.
- Die Extension Member werden dann wie „normale“ Methoden und Properties geschrieben (kein `static`, kein `this`).

Extension Members

- Alte Schreibweise:

```
static class StringExtensions {
    public static string SwitchCase(this string s) {
        StringBuilder sb = new StringBuilder();
        foreach (char c in s) {
            sb.Append(char.IsUpper(c) ? char.ToLower(c) : char.ToUpper(c));
        }
        return sb.ToString();
    }
}
```

statische Klasse

statische Methode, kann aber als Instanz-Methode aufgerufen werden

Schlüsselwort `this` vor dem erstem Parameter, der beim Aufruf nicht mit angegeben wird.

Extension Members

- Neue Schreibweise:

```
static class StringExtensions {
    extension(string s) {
        public string SwitchCase() {
            StringBuilder sb = new StringBuilder();
            foreach (char c in s) {
                sb.Append(char.IsUpper(c) ? char.ToLower(c) : char.ToUpper(c));
            }
            return sb.ToString();
        }
    }
}
```

Immer noch statische Klasse

Erster Parameter (ohne `this`) wandert in den `extension`-Block

Methoden und Properties werden „normal“ geschrieben, also entweder `static` oder nicht, je nach Einsatzzweck.

Extension Members

- Beispiel statische Erweiterungsmethode:

```
static class StringExtensions {
    extension(Console) {
        public static void WriteLineColor(object obj, ConsoleColor color) {
            var oldColor = Console.ForegroundColor;
            Console.ForegroundColor = color;
            Console.WriteLine(obj);
            Console.ForegroundColor = oldColor;
        }
    }
}
```

- Aufruf:

```
static void Main(string[] args) {
    Console.WriteLineColor("Hello World", ConsoleColor.DarkGreen);
    Console.WriteLineColor("And once again", ConsoleColor.Yellow);
}
```

```
Hello World
And once again
```

Extension Members

- Beispiel Instanz-Property:

```
static class StringExtensions {
    extension(string target) {
        public bool IsAscii => target.All(x => char.IsAscii(x));
    }
}
```

- Aufruf:

```
static void Main(string[] args) {
    Console.WriteLine("Hello".IsAscii);
    Console.WriteLine("Hello 🤪".IsAscii);
}
```

```
True
False
```

Extension Members

- Beispiel statisches Property:

```
static class DecimalExtensions {
    extension(decimal) {
        public static decimal Pi => 3.14159265358979323846264338327m;
    }
}
```

- Aufruf:

```
static void Main(string[] args) {
    Console.WriteLine(decimal.Pi);
}
```

```
3,14159265358979323846264338327
```

Vererbung

- Wird eine Basisklasse um Extension Member erweitert, so haben auch die davon abgeleiteten Subklassen diese Member.
- Mit Extension Membern lassen sich sogar als `sealed` gekennzeichnete Klassen erweitern, von denen man keine anderen Klassen ableiten kann.
- Existiert in einer Subklasse bereits ein Member mit gleichem Namen und gleicher Signatur, so hat dieser Vorrang vor dem Extension Member.
- Gibt es den gleichen Extension Member für die Basis- und die Subklasse, so hat die Version der Subklasse Vorrang.

Vererbung 1

```
class PKW {

}

class Cabrio : PKW {

}
```

```
static class ExtensionMethods {
    public static void MethodeA(this PKW p) {
        Console.WriteLine("PKW-ExtMeth A");
    }
}
```

```
static void Main(string[] args) {
    PKW p = new PKW();
    Cabrio c = new Cabrio();

    p.MethodeA();
    c.MethodeA();
}
```

- Es gibt eine Klasse `PKW` und eine Klasse `Cabrio`.
- `Cabrio` erbt von `PKW`.
- Für die Klasse `PKW` gibt es eine Erweiterungsmethode `MethodeA()`.
- Sowohl auf `PKW`- als auch auf `Cabrio`-Objekten kann die `MethodeA()` aufgerufen werden.

Vererbung 2

```
class PKW {
}
class Cabrio : PKW {
    public void MethodeB() {
        Console.WriteLine("Cabrio-Original B");
    }
}
```

```
static class ExtensionMethods {
    public static void MethodeB(this PKW p) {
        Console.WriteLine("PKW-ExtMeth B");
    }
}
```

```
static void Main(string[] args) {
    PKW p = new PKW();
    Cabrio c = new Cabrio();

    p.MethodeB();
    c.MethodeB();
}
```

- Die Klasse `Cabrio` erbt wieder von der Klasse `PKW`.
- Die Klasse `Cabrio` hat eine `MethodeB()`.
- Für die Klasse `PKW` gibt es eine Erweiterungsmethode `MethodeB()`.
- Beim Aufruf von `MethodeB()` auf dem `Cabrio`-Objekt hat die Instanzmethode Vorrang.

Vererbung 3

```
class PKW {
}
class Cabrio : PKW {
}
```

```
static class ExtensionMethods {
    public static void MethodeC(this PKW p) {
        Console.WriteLine("PKW-ExtMeth C");
    }
    public static void MethodeC(this Cabrio c)
        Console.WriteLine("Cabrio-ExtMeth C");
    }
}
```

```
static void Main(string[] args) {
    PKW p = new PKW();
    Cabrio c = new Cabrio();

    p.MethodeC();
    c.MethodeC();
}
```

- Die Klasse `Cabrio` erbt wieder von der Klasse `PKW`.
- Für beide Klassen gibt es jeweils eine eigene Erweiterungsmethode `MethodeC()`.
- Beim Aufruf von `MethodeC()` auf dem `Cabrio`-Objekt hat die spezifischere Methode Vorrang.

Interfaces

- Nicht nur bestehende Klassen können um Extension Member erweitert werden, sondern auch Interfaces.
- Obwohl Interfaces eigentlich nur Methoden-Signaturen vorgeben, können komplette Erweiterungsmethoden mit Methoden-Rumpf für Interfaces erstellt werden und auch Extension Properties.
- Eine Klasse, die das Interface implementiert, hat dann auch die zugehörigen Extension Member.
- Diese Extension Member muss die Klasse nicht, wie bei einem Interface sonst üblich, selbst bereitstellen.

Interfaces

```
public interface IInterface {
    void MethodeA();
}
```

```
static class ExtensionMethods {
    public static void MethodeB(this IInterface i) {
        Console.WriteLine("Interface-ExtMeth");
    }
}
```

```
class ClassA : IInterface {
    public void MethodeA() {
        Console.WriteLine("Wegen IInterface ...");
    }
}
```

```
static void Main(string[] args) {
    ClassA a = new ClassA();
    a.MethodeA();
    a.MethodeB();
}
```

- Ein Interface mit einer `MethodeA()` wird erstellt.
- Das Interface wird um eine `MethodeB()` erweitert.
- Eine Klasse implementiert das Interface und muss `MethodeA()` anbieten.
- Durch die Implementierung des Interface ist jetzt auch `MethodeB()` verfügbar.