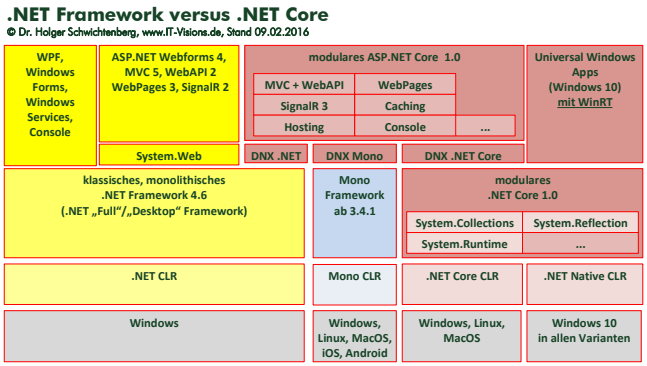
Spickzettel („Cheat Sheet“) Neuerungen in C# Version 6.0 (C# 2015)

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) V1.1 / 09.02.2016

C# 6.0

Die erweiterte Sprachsyntax von C# Version 6.0 ist sowohl in .NET Framework 4.6 als auch .NET Core 1.0 und ab Mono 4.0 verfügbar.



Initialisierung für automatische Properties

Automatische Properties können nun schon bei der Deklaration durch einen Ausdruck initialisiert werden. Der Wert wird in das zugehörige, unsichtbare Feld geschrieben.

```
public class Kunde
 {
  public string Land { get; set; } = "D";
  public DateTime ErzeugtAm { get; set;  } = DateTime.Now;
}
```

Automatische Properties ohne Setter

Automatische Properties, die nur einen Getter haben (Read-only Automatic Properties), darf man bei der Deklaration initialisieren oder auch noch im Konstruktor setzen. Danach ist eine Veränderung nicht mehr möglich. Sie sind dann „immutable“ (unveränderbar).

```
public class Kunde
 {
  public DateTime ErzeugtAm { get; } = DateTime.Now;
  public Kunde(DateTime erzeugtAm)
  {
   // Getter Only Auto Property im Konstruktor setzen
   ErzeugtAm = DateTime.Now;
  }}
```

Expression-bodied Members

Methoden und Read-Only-Properties, die nur einen Ausdruck zurückliefern, kann man nun verkürzt unter Einsatz des Lambda-Operators => schreiben.

```
public string GanzerName => this.Vorname + " " + this.Name;
public decimal NeuerEinkauf(decimal wert) => this.Umsatz += wert;
public override string ToString() => this.GanzerName + ": " + this.Status;
```

Bei *void*-Methoden ist genau ein Statement möglich.

```
public void Print() => Console.WriteLine(this.GanzerName);
```

Operator ?.

Der bisher vorhandene Punkt-Operator (.) liefert einen Laufzeitfehler (*NullReferenceException*), wenn der Ausdruck vor dem Punkt den Wert *null* hat. Der neue Operator *?.* liefert in diesem Fall *null*. Microsoft nennt den Operator Null-conditional Operator oder Null-propagating Operator.

```
string kundenname = repository?.GetKunde()?.Name;
```

Der neue Operator kann auch beim Zugriff auf Indexer eingesetzt werden. Allerdings fängt der Operator nur den Fall ab, dass die Listenvariable den Wert null hat. Es gibt weiterhin einen Laufzeitfehler (*ArgumentOutOfRangeException*), falls es das angesprochene Element in der Liste nicht gibt.

```
string kundenname = kundenListe?[123]?.GanzerName;
```

Operator *nameof*

Der neue Operator *nameof* liefert den Namen eines Bezeichners als Zeichenkette (bei mehrgliederigen Namen nur den letzten Teil). Dieser Operator erhöht die Robustheit und erleichtert das Refactoring in Situationen, in denen der Name einer Klasse oder eines Klassenmitglieds als Zeichenkette zu übergeben ist, z.B. *ArgumentNullException*, Dependency Properties und *PropertyChangedEventArgs*.

```
public void SaveKunde(Kunde kunde)
{
 if (kunde == null) throw new ArgumentNullException(nameof(kunde));
 // ...
}

public int MitarbeiterAnzahl
{
 get { return mitarbeiterAnzahl; }
 set
 {
  PropertyChanged(this, new PropertyChangedEventArgs(nameof(MitarbeiterAnzahl)));
  mitarbeiterAnzahl = value;
 }
}
```

String Interpolation

Durch Voranstellen des Dollarzeichens *$* vor eine Zeichenkette kann man anstelle der nachgestellten Parameterliste bei *String.Format()* die einzelne Ausdrücke direkt in die geschweiften Klammern schreiben. Formatierungsausdrücke sind weiterhin möglich.

```
var ausgabeAlt = String.Format("Kunde #{0:0000}: {1} ist in der Liste seit {2:d}.", k.ID, k.GanzerName, k.ErzeugtAm);
var ausgabeNeu = $"Kunde #{k.ID:0000}: {k.GanzerName} ist in der Liste seit {k.ErzeugtAm:d}.";
```

Bei Verwendung des bedingten Operators *? :* sind zusätzliche runde Klammern vor und nach dem Ausdruck notwendig, um eine Verwechselung mit einem Formatierungsausdruck zu vermeiden.

```
Console.WriteLine($"Stammkunde: {(stammkunde ? "Ja" : "Nein")}");
```

Auch möglich in Verbindung mit @:

```
$@"Pfad: {wurzel}\{ordner}"
```

nicht @$

Spickzettel („Cheat Sheet“) Neuerungen in C# Version 6.0 (C# 2015)

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de) V1.1 / 09.02.2016

Exception Filter

Mit einem Exception Filter (Schlüsselwort *when*) kann man zusätzlich zu den *Exception*-Klassen in den *catch*-Blöcken weiter zwischen verschiedenen Fällen differenzieren.

```
try
{
 var datei = System.IO.File.ReadLines(filename);
}
catch (ArgumentException) when (filename == "")
{
 Console.WriteLine("Ohne Dateiname macht diese Aktion
keinen Sinn!");
}
catch (ArgumentException ex) when
(ex.Message.Contains("Illegales"))
{
 Console.WriteLine("Ungültige Zeichen im Dateinamen: " +
filename);
}
catch (ArgumentException ex)
{
 Console.WriteLine("Ungültige Angabe: " + filename + ":" +
ex.Message);
}
catch (NotSupportedException ex) when
(ex.Message.Contains("format"))
{
 Console.WriteLine("Ungültiges Format!");
}
catch (NotSupportedException ex)
{
 Console.WriteLine("Nicht unterstützt: " + ex.Message);
}
 catch (FileNotFoundException ex)
{
 Console.WriteLine("Datei " + filename + " nicht gefunden");
}
catch (Exception ex)
{
 Console.WriteLine("Anderer Fehler: " + ex.Message);
}
```

using für statische Mitglieder

Im *using*-Block einer C#-Programmcodedatei kann man nun statische Klassen einbinden:

```
using static System.Console;
using static System.Environment;
```

Danach darf man die statischen Mitglieder dieser Klassen ohne Nennung des Klassennamens aufrufen.

```
// bisherige Schreibweise
Console.WriteLine(Environment.UserDomainName + @"\" +
Environment.UserName);

// neu in C# 6.0
WriteLine(UserDomainName + @"\" + UserName);
```

Index Initializer

Für die Initialisierung von *Dictionary*-Objekten bietet C# 6.0 eine zusätzliche Schreibweise an.

```
// bisherige Dictionary Initializer (seit C# 3.0)
var PLZListe1 = new Dictionary<int, string>()
{
  { 45127, "Essen" },
  { 30625, "Hannover" },
  { 69123, "Heidelberg" }
};

// neue Dictionary Initializer (ab C# 6.0)
var PLZListe2 = new Dictionary<int, string>
{
 [45127] = "Essen",
 [30625] = "Hannover",
 [69123] = "Heidelberg"
};
```

await in catch- und finally-Blöcken

Die Verwendung des Schlüsselwortes *await* ist nun auch in *catch*- und *finally*-Blöcken erlaubt.

```
try
{
 conn = new SqlConnection(CONNSTRING);
 await conn.OpenAsync();
 var cmd = new SqlCommand(SQL, conn);
 var reader = await cmd.ExecuteReaderAsync();
 anzahl = reader.FieldCount;
 Console.WriteLine("Anzahl geladener Spalten: " + anzahl);
 return anzahl;
}
catch (Exception ex)
{
 var errorID = await LogErrorAsync(ex);
 Console.WriteLine("Fehler wurde unter Nummer " + errorID
+ " protokolliert");
 return -1;
}
finally
{
 if (conn != null) conn.Close();
 var wc = new WebClient();
 await
wc.DownloadStringTaskAsync("http://server/logservice/" +
anzahl);
}
```

Links

C# 6.0 ist Teil der .NET Compiler Platform “Roslyn”
https://github.com/dotnet/roslyn/

Über den Autor

Dr. Holger Schwichtenberg gehört zu den bekanntesten Experten für die Programmierung mit Microsoft-Produkten in Deutschland. Er hat zahlreiche Bücher zu .NET und PowerShell veröffentlicht und spricht regelmäßig auf Fachkonferenzen wie der BASTA. Sie können ihn für Schulungen, Beratungen und Projekte buchen.

E-Mail: hs@IT-Visions.de
Website: www.IT-Visions.de
Weblog: www.dotnet-doktor.de.