C# 9.0

Der große Wurf

Von Records, Top-Level Statements und Pattern Matching.

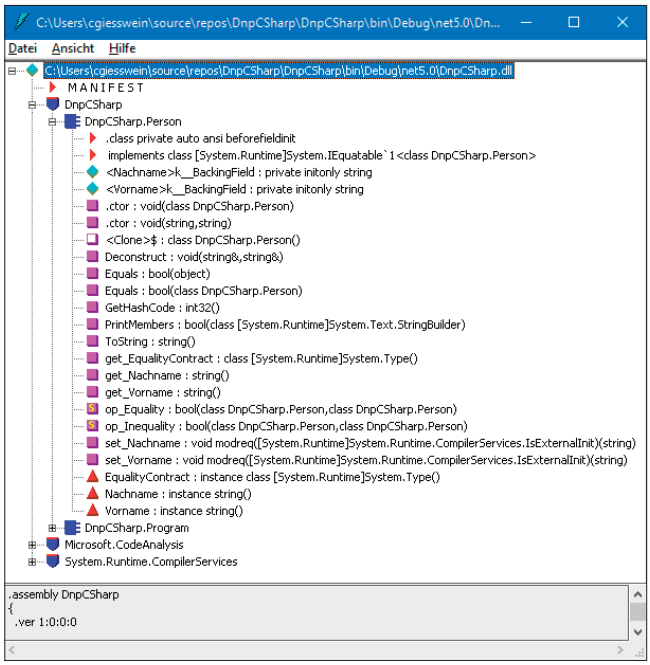
C# 9.0 ist endlich da – der jüngste Spross aus der Reihe der Sprachversionen von C#. Er wurde im November 2020 mit .NET 5.0 veröffentlicht. Erneut ist der Bruch mit dem .NET Framework da, neue Features sind hinzugekommen und es gibt ein paar sehr interessante neue Schlüsselwörter. Aber der Reihe nach.

Voraussetzungen

Die Voraussetzungen für C# 9.0 sind nicht unerheblich. Denn im Gegensatz zu C# 8.0, für das .NET Standard 2.1 Voraussetzung war, ist C# 9.0 nun nur noch mit .NET 5 kompatibel. .NET Standard war und ist lange die Brücke zwischen .NET Framework und .NET Core gewesen. Mit .NET 5 schneidet Microsoft immer mehr Zöpfe ab und .NET Standard wird weiter in den Hintergrund gedrängt. Das macht sich auch in der Sprache bemerkbar. Daher gilt: Wer mit den neuesten Features arbeiten möchte, kommt um .NET 5 nicht herum.

Records – mein neues Lieblingsfeature

Mit jeder Version verfolgt Microsoft gewisse Ziele. Bei C# 9.0 ist es das Thema Datengestaltung. C# 9.0 führt das neue Schlüsselwort *record* ein, das für die Modellierung einer Datenklasse verwendet werden kann. Records sind dabei ganz normale Klassen, wobei der Compiler sehr viel vom Klassenaufbau automatisch und implizit generiert.



Das Innenleben des Records Person im IL-Disassembler (Bild 1)

Aber fangen wir mit der einfachsten, doch gleichzeitig mächtigsten Zeile an und zerlegen sie anschließend in gewohnter Weise bis auf den IL-Code:

```
record Person(string Vorname, string Nachname);
```

Mit diesem Konstrukt haben Sie bereits die kürzeste Schreibweise eines Records vor Augen. Damit erhalten Sie eine Klasse mit relativ viel Fleisch – allem voran zwei Properties: *Vorname* und *Nachname*. Gleichzeitig erzeugt der Compiler aber auch einen Konstruktor, deshalb lässt die Verwendung des Records *Person* nun folgenden Code zu:

```
Person p = new Person("Christian", "Giesswein");
```

Wer jedoch den Eigenschaften später neue Werte zuweisen will, wird enttäuscht: Die Properties lassen sich nachträglich nicht mehr ändern – ähnlich den anonymen Typen, die schon seit mehreren Jahren zur Verfügung stehen:

```
Person p = new Person("Christian", "Giesswein");
p.Vorname = null;
  // CS8852: Init Only Eigenschaft
var x = new { Vorname = "Christian" };
x.Vorname = null;
  // CS0200: Schreibgeschützte Eigenschaft
```

Sie erhalten hier zwei unterschiedliche Fehlermeldungen, wenn Sie den Properties neue Werte zuweisen wollen: Im Fall der anonymen Klasse *CS0200*: Die Eigenschaft ist schreibgeschützt. Im Fall des Records *CS8852*: Es handelt sich um eine Init-only-Eigenschaft – ebenfalls ein neues Konzept von C# 9.0. Aber dazu später mehr.

Ruft man nun wiederum den IL-Disassembler auf und zerlegt dort die Anwendung, so erhält man eine große Menge an generierten Anweisungen (Bild 1), wobei auffällt:

- Ein Record ist eine normale Klasse.
- Es sind zwei Konstruktoren definiert: einer für zwei Strings als Parameter und ein weiterer, dem eine andere *Person* übergeben werden kann.
- Die Properties sind in gewohnter Weise als Backingfields ausgeführt, die aber mit dem Zusatz *initonly* versehen sind.
- Es gibt eine Methode *PrintMembers*, dem ein *StringBuilder* übergeben werden kann.
- Die Klasse implementiert *IEquatable<Person>*.
- Eine versteckte *Clone*-Methode ist vorhanden.
- *GetHashCode*, *Equals* und die Vergleichsoperatoren „==“ sowie „!=“ sind überschrieben.

Mit einem Record führt Microsoft einen sogenannten Immutable-Datentyp (unveränderlich) ein. Immutable insofern, da keine direkte Eigenschaft geändert werden kann. Damit lassen sich also Datentypen nun sehr kurz und komfortabel schreiben.

Gerade für Anwendungen, die mit vielen Datenklassen arbeiten, ist dies eine enorme Erleichterung – deswegen auch: mein neues Lieblingsfeature.

Die etwas komplizierteren Fälle

Ein Record lässt sich wie gezeigt sehr einfach definieren. Es steht aber auch eine ausführlichere Schreibweise zur Verfügung:

```
record Person
{
  public string Vorname { get; init; }
  public string Nachname { get; init; }
}
```

Ein Record kann also sehr ähnlich zur Klasse auch mit normalen Properties versehen werden. Bei automatischen Properties gibt es nun neben *get* und *set* eine dritte Möglichkeit: *init*. Das bedeutet, diese Eigenschaft darf nur in der sogenannten Initialisierungsphase gesetzt werden – also im Konstruktor oder im Object Initializer. Deshalb ist in diesem Fall die folgende Schreibweise gültig:

```
Person p = new Person
{
  Vorname = "Christian",
  Nachname = "Giesswein"
};
p.Vorname = null; // CS8852 - Init Only Eigenschaft
```

Übrigens: Das Schlüsselwort *init* kann auch in Klassen verwendet werden – schließlich ist ein Record ja auch nichts anderes.

OOP mit Records? Nur zu!

Das bedeutet aber auch, dass die objektorientierten Aspekte von C# auch für Records gelten: Ein Record darf von anderen Typen (Records) ableiten oder Interfaces implementieren.

```
record Person : IDisposable
{
  public string Vorname
    { get; init; }
  public string Nachname
    { get; init; }

  public void Dispose()
  {
  }
}
```

```
record OtherPerson : Person;
record OtherOtherPerson(string MiddleName)
  : OtherPerson;

record OtherPerson2 : MyClass;
  //CS8864 - Nur von Records sind Ableitungen möglich
class MyClass { }
```

Wie im Beispielcode ersichtlich, wird das Interface *IDisposable* in dem Record ganz normal implementiert. Auch leitet sich der Record *OtherPerson* vom Record *Person* ab. Ein weiterer Record *OtherOtherPerson* definiert einen Konstruktor, bei dem der *Middlename* angegeben werden muss. Was nicht funktioniert, ist, eine Klasse zu erstellen, die von einem Record ableitet. Dies verbietet der Compiler.

Der with-Operator

Was hat es mit dem Immutable auf sich? Am Anfang wäre da die Idee, dass sich ein einmal instanziertes Objekt nicht mehr verändern lässt, sondern dass jede Änderung zu einer neuen Instanz führt. Und genau dies unterstützen die Records automatisch. Das ist auch der Grund, warum automatisch ein Konstruktor erzeugt wird, der als Copy-Constructor bekannt ist. Bei einer Änderung wird das ursprüngliche Objekt einfach kopiert und die veränderten Werte werden übernommen.

Dafür hat sich Microsoft für C# 9.0 den *with*-Operator einfallen lassen:

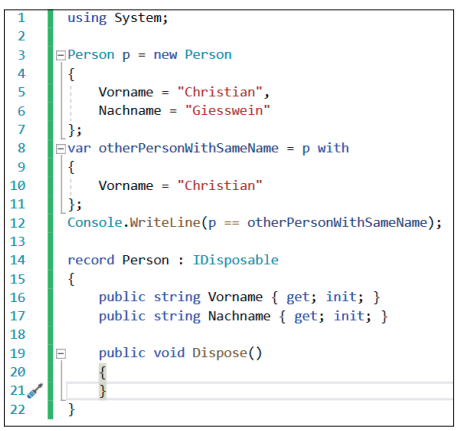
```
Person p = new Person
{
  Vorname = "Christian",
  Nachname = "Giesswein"
};
Person otherPerson = p with
{
  Vorname = "Other"
};
```

Mit dem *with*-Operator wird angegeben, welche Properties einen anderen Wert erhalten sollen. Somit hat die Variable *otherPerson* denselben Nachnamen aber eben einen anderen Vornamen. Im Hintergrund wird dabei der Copy-Constructor aufgerufen.

Wertgleichheit!

Wir haben im IL-Code einige Member für die Vergleichsmethoden von .NET entdeckt. Um zu prüfen, was es damit auf sich hat, ändern wir den Beispielcode etwas ab:

```
Person p = new Person
{
  Vorname = "Christian",
  Nachname = "Giesswein"
};
```



Top-Level Statements: Namespace und Klasse können entfallen (Bild 2)

```
var otherPersonWithSameName
  = p with
{
  Vorname = "Christian"
};
Console.WriteLine(
  p == otherPersonWithSameName);
```

Die letzte Zeile vergleicht die beiden Records mit dem ==-Operator. Bei normalen Klassen muss man davon ausgehen, dass dieser Vergleich *false* ergibt, da für gewöhnlich die Referenz überprüft wird.

Bei Records vergleichen die Methoden hingegen die Inhalte. Deshalb spricht man auch von Wertgleichheit. In unserem Beispiel wird auf der Konsole *true* ausgegeben, da alle Properties denselben Wert haben.

Top-Level Statements

Eine weitere Neuerung, die wahrscheinlich in großen Projekten eher unpassend erscheint, ist das Feature „Top-Level Statements". C# 9.0 erlaubt es, dass es in einem Projekt eine einzige Datei gibt, die anders aufgebaut ist – sie darf direkt mit Statements beginnen. Das bedeutet, wir könnten unser Beispiel wie in Bild 2 ersichtlich schreiben.

Einzige Regel hierbei: Typdefinitionen müssen ans Ende rutschen (wie unser Record in Zeile 14). Im Hintergrund wird dabei eine Klasse *Program* mit einer statischen *Main*-Methode definiert (Bild 3) – also ein recht billiger Trick, der aber für Beispielcode oder Codeschnipsel unnötige Arbeit erspart.

Geht's noch kürzer?

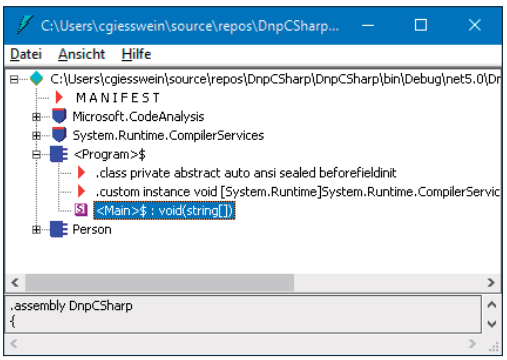
Ein Feature, das etwas länger gedauert hat, bis es implementiert war, ist der neue *new*-Operator. Ähnlich wie bei der Verwendung von *var* wird vom Compiler erkannt, was der Zieltyp ist. Somit kann man sich die Angabe des Typs hinter *new* sparen. Unser Beispiel lässt sich noch weiter verkürzen:

```
Person p = new()
{
  Vorname = "Christian",
  Nachname = "Giesswein"
};
```

Deshalb kann man nun auch Properties direkt instanzieren:

```
record Person : IDisposable
{
  public List<Person> Children { get; } = new();
}
```

Hier sieht man übrigens auch die Grenze der Immutable Records: Es wird nicht verhindert, dass jemand den Inhalt der Eigenschaft *Children* ändert. Also können munter Einträge hinzugefügt oder auch gelöscht werden. Hier gilt es also aufzupassen.



Der Compiler erzeugt die Klasse Program und die Main-Methode im Hintergrund (Bild 3)

Pattern Matching – Folge 42

Auch das Pattern Matching wurde in C# 9.0 erweitert. So halten nun Vergleichsoperatoren und Verknüpfungen Einzug. VB.NET lässt grüßen. Es lassen sich folgende Vergleiche bewerkstelligen:

```
int second = DateTime.Now.Second;
if(second is > 3 and < 10 or 42
  and not 43)
{
}
```

Die *if*-Klammer wird nur ausgeführt, wenn die Sekunde größer drei und kleiner zehn oder gleich 42 und auf keinen Fall 43 ist. Beachten Sie: Der *is*-Operator darf nur einmal vorkommen. Zugelassen sind dabei die Vergleichsoperatoren sowie Konstanten. Was mit C# 9.0 sicherlich angenehmer zu lesen ist, ist folgende Variante eines Ist-null-Checks:

```
string s = null;
if(s is not null) { }
```

Dies hat durchaus seinen Charme, liest sich aber bestimmt auf den ersten Blick etwas ungewohnt. Im Netz heftig diskutiert wird eine Variante, um sich den Methodenaufruf von *string.IsNullOrEmpty(...)* zu sparen:

```
string s = null;
if(s is { Length: > 0 }) { }
```

Hier sollte jeder für sich überlegen, ob diese Schreibweise wirklich besser lesbar ist.

Fazit

Wir dürfen uns nun jedes Jahr auf ein neues Update der Sprache C# freuen. Aber gerade die Version 9 führt meines Erachtens (endlich) sehr nützliche Dinge in die Sprache ein. Oft wird der Record-Datentyp etwas kleingeredet. Zerlegt man diesen jedoch in seine Einzelteile und erkennt die notwendigen Sprachkonstrukte, wirkt er überzeugend. So lassen sich mit der Kombination aus neuen und alten C#-Konstrukten geniale Dinge lösen. Für mich eine sehr spannende Version, die Microsoft hier herausgebracht hat – weiter so! ■

Christian Giesswein
studierte Wirtschaftsinformatik in Wien und entwickelt seit klein auf Software mit .NET und C#. In Tirol hat er das Unternehmen Giesswein Software-Solutions gegründet, das sich auf Individualsoftware und Consulting spezialisiert.
christian@software.tirol

dnpCode A2101NETirol