KOCHEN

mit PATRICK A. LORENZ

HÄTTEN SIE ES GEWUSST?

Unnatürliche Auslese

Fragen und Antworten aus Bewerbungsgesprächen für .NET-Entwickler.

Exakt 60 Bewerbungsgespräche habe ich 2016 absolviert – ich habe nachgezählt. Die Einladung zu einem ersten persönlichen Gespräch ging an knapp 20 Prozent der Bewerber. Darunter waren zukünftige Auszubildende als Fachinformatiker Anwendungsentwicklung und Systemintegration, Hochschul-Praktikanten, Anwärter auf kaufmännische Positionen, vor allem aber gelernte Softwareentwickler und -entwicklerinnen mit Berufserfahrung. Der Schwerpunkt unserer Suche lag und liegt darin, Entwickler zu finden, die uns in den Bereichen Angular, Web-Frontend allgemein sowie bei der serverseitigen Entwicklung im Microsoft-Umfeld mit .NET und C# unterstützen.

Jedes Bewerbungsgespräch ist anders, jeder Lebenslauf ist anders, jede Persönlichkeit ist individuell und muss ebenso individuell betrachtet werden. Wir sind nett zu unseren Bewerbern, begegnen uns auf Augenhöhe. Wenn die Chemie stimmt, wird viel gelacht. Wenn es menschlich oder fachlich nicht passt, geben wir jedem, der es hören möchte, unseren Eindruck in konstruktiver Form für sein nächstes Gespräch mit. Trotzdem bleibt ein Bewerbungsgespräch eine gegenseitige Bewertung, die wiederkehrenden Kriterien folgt. Auf unserer Seite versuchen wir festzustellen, ob unser Gegenüber in unser Team-Gefüge passt. Und wir versuchen zu antizipieren, ob wir dem potenziellen Mitarbeiter das bieten können, was er sucht, um sich über Jahre im Unternehmen wohlzufühlen und sich entwickeln zu können.

Mitunter einfacher als die Einschätzung der persönlichen Skills ist eine fachliche Bewertung. Wir legen dabei kein starres Raster basierend auf der Ausbildung an. Ein abgebrochenes Studium ist kein Ausschlussgrund. Wichtiger als das, was der Bewerber heute schon weiß, ist die Einschätzung, was er lernen kann. Welches Potenzial kann der neue Mitarbeiter einbringen, wie kann er sich weiterentwickeln? Und was für Grundlagenwissen bringt er mit, um wichtige Querverbindungen herstellen zu können?

.NET-Entwickler mit Berufserfahrung überraschen wir je nach Position mit einigen sehr konkreten technischen Fragen. Aus den Antworten lässt sich viel ableiten, nicht nur das konkrete Wissen, sondern eben auch Grundlagen und die Fähigkeit des Bewerbers, Antworten herzuleiten. Einige unserer Fragen aus den letzten Monaten möchte ich Ihnen heute vorstellen. Die Antworten sind ein Anhaltspunkt dafür, was wir erwartet haben, andere Antworten können aber ebenso gut oder besser sein.

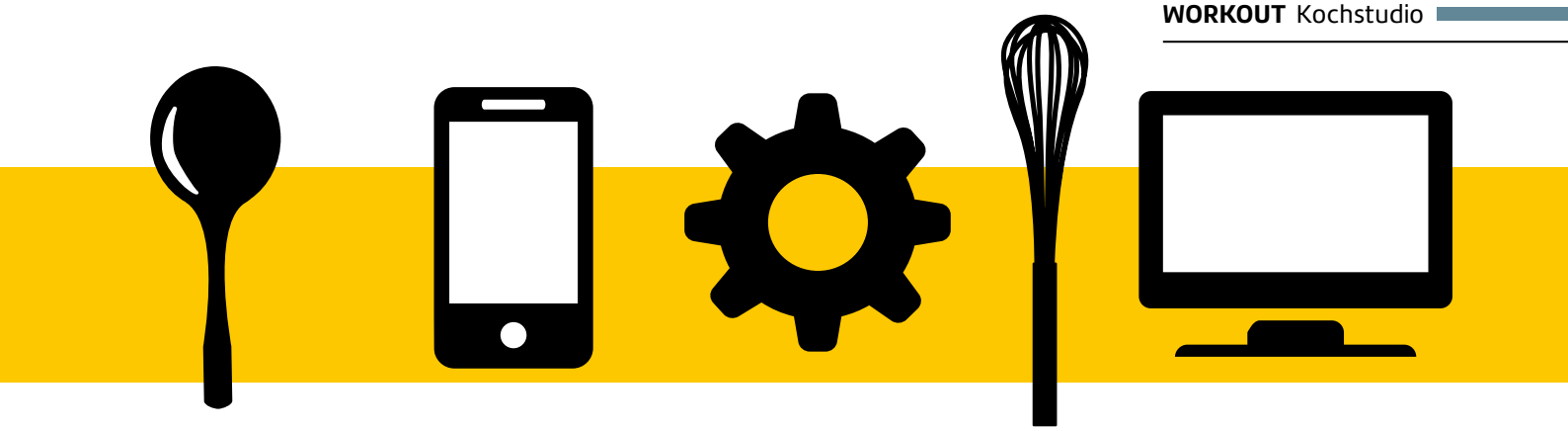
Frage: *Was ist der Unterschied zwischen den Datentypen string und String?*
Antwort: *String* ist die Klasse *System.String*, ein nativer Datentyp des .NET Frameworks, der in allen .NET-Sprachen und der Intermediate Language genutzt wird. *string* ist der C#-Alias für den Datentyp. Technisch besteht kein Unterschied. C# kennt spezifische Alias-Namen für die allermeisten primitiven .NET-Datentypen, etwa *int* für *System.Int32*, *bool* für *System.Boolean* und so weiter.

Frage: *Wie lässt sich die Klasse String funktional erweitern?*
Antwort: Die Klasse *System.String* ist als *sealed* gekennzeichnet und kann deshalb nicht abgeleitet werden. Zusätzliche Funktionalität lässt sich in Form von Erweiterungsmethoden bereitstellen:

```
public static class StringExtensions {
  public static bool IsNotEmpty(this string value)
  {
    return (string.IsNullOrEmpty(value) == false);
  }
}
var hello = "Hello";
if (hello.IsNotEmpty()) { }
```

Frage: *Warum ist das folgende Snippet nicht empfehlenswert, und wie geht es besser?*

```
var str = "Hallo " + salutation + " " + lastname + ",";
```



Antwort: *String* ist immutable, also ein unveränderbarer Datentyp. Eine Zeichenkette kann daher nicht verändert werden. Stattdessen wird bei einer Änderung eine neue Zeichenkette erzeugt. Im Beispiel existieren zwei Variablen und drei statische Zeichenketten. Durch die Verkettung werden fünf weitere Zeichenketten erzeugt, wobei nur die letzte benötigt wird.

Es gibt verschiedene bessere Alternativen, darunter die Klasse *StringBuilder* sowie die Methoden *string.Concat* und *string.Format*:

```
var str =
  string.Format("Hallo {0} {1},", salutation, name);
```

Mit der in C# 6.0 eingeführten String Interpolation geht es noch eleganter:

```
var str = $"Hallo {saluation} {name},";
```

Frage: *Was ist der Unterschied zwischen var und object?*
Antwort: *object* ist der oberste Basisdatentyp. Jedes Objekt lässt sich per Polymorphie dem Datentyp *object* zuweisen. Eine Variable vom Typ *object* gilt als untypisiert. Spezifische Mitglieder des tatsächlichen Datentyps lassen sich nicht aufrufen, hier etwa die Eigenschaft *Length*:

```
object str = "Hallo Welt";
var length = str.Length; // Compiler-Fehler
```

var ist kein echter Datentyp, sondern lediglich eine Komfortfunktion des Compilers. Statt den Datentyp explizit zu notieren, erkennt der Compiler diesen aus der Zuweisung. Technisch macht es also keinen Unterschied, ob man nun *string* oder *var* notiert:

```
var str = "Hallo Welt";
var length = str.Length;
```

var wurde mit C# 3.0 eingeführt, um unter anderem die Zuweisung von anonymen Datentypen zu ermöglichen.

Frage: *Was ist der Null Coalescing Operator und was der Null Conditional Operator?*
Antwort: Der Null Coalescing Operator existiert seit der ersten Version von C#. Er wird durch ein doppeltes Fragezeichen repräsentiert. Der Operator prüft den linken Operanden auf *null*. Ist dieser nicht *null*, so wird er zurückgeliefert, ansonsten der rechte Operand.

```
var z = x ?? y;

// Langform:
var z = (x != null ? x : y)
```

Der Null Conditional Operator wurde mit C# 6.0 neu eingeführt. Er kann beim Zugriff auf Referenzen zur impliziten *null*-Prüfung genutzt werden. Der Operator wird als Fragezeichen mit darauffolgendem Punkt notiert:

```
var str = null;
var length = str?.Length;
```

Ohne das Fragezeichen würde der Zugriff auf die Eigenschaft *Length* zu einer NullReferenceException führen. Der Null Conditional Operator prüft den linken Operanden auf *null* und führt den rechts folgenden Ausdruck gegebenenfalls gar nicht mehr aus, sondern liefert *null* zurück. Die Variable *length* ist folglich vom Typ *int?,* also *Nullable<int>*. Der Operator funktioniert auch in Verbindung mit Methoden:

```
var clone = str?.Clone();
```

Beide Operatoren sind der Kategorie „syntactic sugar" zuzuordnen, im Alltag aber ungemein praktisch. Gerade der Null Conditional Operator kann in der Praxis häufig genutzt werden, um Code übersichtlicher zu gestalten, und ist bereits zu einem sehr deutlich wahrnehmbaren Sprachbestandteil geworden.

Frage: *Was ist der Unterschied zwischen den Klassen-Modifiern abstract, sealed und static?*
Antwort: Eine Klasse mit *abstract* muss abgeleitet werden und kann selbst nicht instanziert werden. Das Schlüsselwort wird oft für Basisklassen verwendet.

sealed verhindert, dass eine Klasse abgeleitet wird. Die Klasse kann in aller Regel instanziert werden, es sei denn, private Konstruktoren verhindern dies. Ein bekanntes Beispiel für *sealed* ist die Klasse *string,* die keine Ableitungen erlaubt.

static kennzeichnet eine statische Klasse, die somit nicht instanziert werden kann. Auch eine Ableitung ist nicht möglich. Das Schlüsselwort ist Voraussetzung für Klassen, die Erweiterungsmethoden zur Verfügung stellen. ▶

Alle drei Schlüsselwörter schließen sich gegenseitig aus, eine Klasse kann nur als *abstract* oder *sealed* oder *static* gekennzeichnet werden.

Frage: *Was ist Unterschied zwischen abstrakten Klassen und Schnittstellen?*
Antwort: Eine abstrakte Klasse kann im gleichen Umfang Implementierungen enthalten wie eine normale Klasse, einzig kann sie nur durch eine Ableitung instanziert werden.

Eine Schnittstelle enthält keinerlei Implementierungen, sondern definiert lediglich einen Vertrag der zu implementierenden öffentlichen Klassenmitglieder und deren Signatur.

Frage: *Mit welchem Schlüsselwort unterstützt C# Mehrfachvererbung?*
Antwort: Fangfrage! C# unterstützt die Mehrfachvererbung nicht. Klassen können nur von einer Basisklasse erben, können aber mehrere Schnittstellen implementieren.

Frage: *Wozu wird die Schnittstelle IDisposable genutzt?*
Antwort: Die Schnittstelle *IDisposable* implementiert die Methode *Dispose*. Die Schnittstelle wird üblicherweise von Klassen implementiert, die intern nicht verwaltete Ressourcen halten, wie etwa Verbindungen zu einer Datenbank oder in das Dateisystem. Diese Ressourcen werden durch den Aufruf von *Dispose* freigegeben.

Die Schnittstelle wird üblicherweise in Verbindung mit einem *using*-Block verwendet:

```
using (var stream = new FileStream(filePath)) {
  ...
}
```

Bei diesem Konstrukt wird sichergestellt, dass die *Dispose*-Methode auch dann aufgerufen wird, wenn der *using*-Block per *return* oder durch eine unerwartet auftretende Exception vorzeitig verlassen wird. Die externen Ressourcen werden dadurch sicher freigegeben.

Frage: *Was ist der Unterschied zwischen using und try..catch..finally?*
Antwort: Es gibt keinen. Der Compiler setzt einen *using*-Block als *try..catch..finally* um:

```
var stream = new FileStream(filePath);
try {
  ...
}
finally {
  if(stream != null) stream.Dispose();
}
```

Beim *using*-Block handelt es sich also um „syntactic sugar", einen Komfort für den Entwickler.

Frage: *Was ist der Unterschied zwischen throw und throw e in einem catch-Block?*

```
try {
  ...
}
catch(Exception e) {
  throw;
  // ... oder ...
  throw e;
}
```

Antwort: *throw* wirft die Exception ohne Änderung mit dem ursprünglichen Stack Trace weiter nach oben. Bei *throw e* wird der Stack Trace beginnend mit dem erneuten Werfen neu aufgebaut. Ein wesentlicher Teil der Information geht somit verloren.

Frage: *Was ist der Unterschied zwischen Werte- und Referenztypen, und welche Beispiele bietet das .NET Framework?*
Antwort Wertetypen:

- Wertetypen habe eine feste Größe und sind damit nicht veränderbar.
- Eine Variable enthält direkt den auf dem Stack abgelegten Wert.
- Wird eine Variable kopiert, wird der Wert kopiert.
- Wertetypen haben immer einen Wert.
- Für jeden Wertetyp ist ein Standardwert vorgesehen, bei numerischen Typen etwa 0.
- Beim Vergleich wird der Wert verglichen.
- Beispiele für Wertetypen sind *int, bool* und andere von *System.ValueType* abgeleitete Datentypen.

Antwort Referenztypen:

- Bei Referenztypen enthält die Variable auf dem Stack nicht den Inhalt, sondern eine Adresse im dynamischen Speicher/Heap, unter der der Inhalt zu finden ist.
- Das Objekt verfügt über eine dynamische Größe.
- Wird ein Referenztyp übergeben, wird nicht das Objekt kopiert, sondern der Verweis. Mehrere Verweise können dieselbe Adresse enthalten und somit dasselbe Objekt.
- Änderungen an Referenzytpen sind möglich und wirken sich auf alle Verweise aus.
- Referenztypen haben keinen Standardwert, sondern nehmen eingangs den Wert null an.
- Beim Vergleich wird der Verweis beziehungsweise die Adresse verglichen, nicht der Inhalt.
- Beispiele für Referenztypen sind sämtliche Klassen des .NET Frameworks.

Diese Frage empfinde ich als sehr wichtig, da sie oftmals die Spreu vom Weizen trennt und Hintergrundinformationen offenbart, die für die Arbeit mit .NET nicht unmittelbar erforderlich sind.

Frage: *Ist String ein Werte- oder Referenztyp?*
Antwort: *String* ist ein Referenztyp, übernimmt aber viele Eigenschaften von Wertetypen und wird daher oft für einen solchen gehalten. Zeichenketten sind nicht veränderbar. Sie werden als Wert übergeben beziehungsweise kopiert. Beim

Vergleich wird der Wert und nicht die Referenz verglichen. Andererseits ist eine Zeichenkette zunächst *null* und verfügt über eine dynamische Größe.

Frage: *Was ist Boxing/Unboxing?*
Antwort: Boxing bezeichnet die Konvertierung eines Wertetyps in einen Referenztyp, üblicherweise *object*. Der Wert *1* wird im Beispiel auf den Heap kopiert, die Variable *o* enthält wie bei Referenztypen einen Verweis. Unboxing kehrt diesen Prozess um:

```
int i = 1;
object o = i;

// unboxing
i = (int) o;
```

Boxing und Unboxing kam in der Praxis vor allem in der Zeit vor Generics vor, wenn Wertetypen in einer generischen Collection wie *ArrayList* gespeichert werden sollten, die ihre Daten intern mit dem Datentyp *object* verwaltet. Die generische Collection *List<T>* verhindert das Boxing. Boxing sollte unbedingt vermieden werden.

Frage: *Wofür steht die Abkürzung GAC, und was ist das?*
Antwort: GAC steht für Global Assembly Cache. Es handelt sich um eine zentrale Ablage für von mehreren Applikationen genutzten Bibliotheken mit dem Ziel, diese nicht für jede Applikation auf dem System als Kopie vorzuhalten. Die Identifikation von Assemblies im GAC erfolgt über einen sogenannten Strong Name, der neben dem Namen vor allem auch die genaue Version enthält. Dadurch ist es möglich, dass im GAC mehrere Versionen einer Bibliothek vorgehalten und parallel genutzt werden können.

Frage: *Wofür steht die Abkürzung GC, und was ist das?*
Antwort: GC steht für den .NET Garbage Collector, der für die Garbage Collection verantwortlich ist. Dabei handelt es sich um einen grundlegenden Mechanismus des .NET-Speichermanagements, der für die Erkennung von nicht mehr benötigten beziehungsweise nicht mehr referenzierten Objekten verantwortlich ist. Der Garbage Collector gibt die von diesen Objekten reservierten Speicherbereiche auf dem verwalteten Heap wieder frei.

Frage: *Was sind Generationen von Objekten?*
Antwort: Das .NET-Speichermanagement gliedert Objekte auf dem verwalteten Heap fest in drei Generationen:

- Generation 0 – Kurzlebige Objekte wie lokale Variablen.
- Generation 1 – Ältere Objekte als Puffer zwischen Generation 0 und 2.
- Generation 2 – Langlebige oder besonders große Objekte.
- Die Generationen dienen einer Optimierung der Garbage Collection, die sich vornehmlich auf Objekte der Generation 0 konzentriert, also beispielsweise lokale Variablen schnell aufräumt. Nur eine komplette, aufwendige Collection umfasst auch Objekte der Generation 2.

Fazit

Das ist nur ein kleiner Auszug, einige wichtige Themen wie OOP, Generics, IEnumerable, Lambda-Ausdrücke und so weiter habe ich aus Platzgründen vernachlässigt. Auch in den Gesprächen sprechen wir nur einen kleinen, hoffentlich passenden Teil der Fragen an. Auf viele gibt es viele falsche und einige mehr oder weniger richtige Antworten. Meine Antworten sind nur eine Variante, ich hoffe aber wenigstens eine richtige. Was es nicht gibt, ist ein Punktesystem oder eine starre Einordnung. Ein Java-Entwickler wird mutmaßlich an C#-Schlüsselwörtern scheitern, dafür vielleicht solides Grundlagenwissen offenbaren. Entsprechend individuell ist unser Entscheidungsprozess, der in einer Zu- oder Absage oder einer Einladung zu einem Probearbeitstag mündet.

Wenn ich an die zurückliegenden Bewerbungsgespräche denke, bin ich immer wieder erstaunt, wie unterschiedlich und interessant Menschen sind. Im Gedächtnis bleiben natürlich vor allem die kuriosen Momente. Da war ein Bewerber, der von jedem vorherigen Arbeitgeber erklärt hat, warum dieser sch**** sei und wer wen ein Ar******ch genannt hat. Die meisten Arbeitgeber habe er anschließend verklagt. Den letzten allerdings nicht, denn seine Rechtsschutzversicherung würde ihm kündigen, wenn er innerhalb eines Jahres zweimal vors Arbeitsgericht ziehen würde.

Ein anderer Bewerber hat in den ersten Sätzen erklärt, er sei extrem faul. Das war dumm, aber es war auch mutig. Er hatte die Wahl, daraus etwas zu machen oder das Gespräch nach wenigen Minuten zu beenden. Und tatsächlich hat er es geschafft, plausibel zu erklären, dass ein fauler Entwickler ein guter Entwickler ist. Im Kern der Aussage stimme ich mit ihm überein, und das anschließende Gespräch war nicht schlecht. Aber selbst wenn inhaltlich richtig, sind derartige Aussagen für ein Bewerbungsgespräch doch eher ungeschickt.

Wir starten jedes Gespräch mit der Frage, was der Bewerber schon von uns als Unternehmen weiß. Viele Gesprächspartner haben sich gut vorbereitet. Viele aber auch nicht. Das hat mich immer wieder erstaunt. Ich kann jedem Bewerber nur ans Herz legen, sich unbedingt gut auf das Gespräch, das Unternehmen und möglichst die konkreten Gegenüber vorzubereiten. Und das darf man selbstverständlich auch andersherum erwarten.

Am meisten beeindruckt haben mich übrigens solche Bewerber, die selbst kreative Fragen mitgebracht und damit auch uns zum Nachdenken gebracht oder einem Test unterzogen haben. Chapeau! ■

Patrick A. Lorenz
ist Geschäftsführer der PGK GmbH, eines auf Microsoft-Technologien spezialisierten Dienstleisters und Anbieter des Content-Management-Systems QualiSite. Daneben ist er als Autor und Coach tätig. In seiner Freizeit ist Patrick Hobbykoch. Sie erreichen ihn unter www.pgk.de, lorenz@pgk.de und auf der Facebook-Seite zur Kolumne unter
http://fb.com/Kochen.mit.Patrick.

dnpCode A1702Kochstudio