Eine Einführung in C#

Maximilian Irlbeck
Lehrstuhl für Software & Systems Engineering

27. Oktober 2011

Inhalt

- 1. Grundlagen
  - Historie
  - Allgemeine Infos zu C#
- 2. Sprachdesign
- 3. Sprachfeatures/-syntax

Historie: C# war Cool.

- ***1990‘s***
  Erste Klassen für „*Next Generation Windows Services*“ werden in **SMC** verfasst
  (**S**imple **M**anaged **C** , „*smack*“)

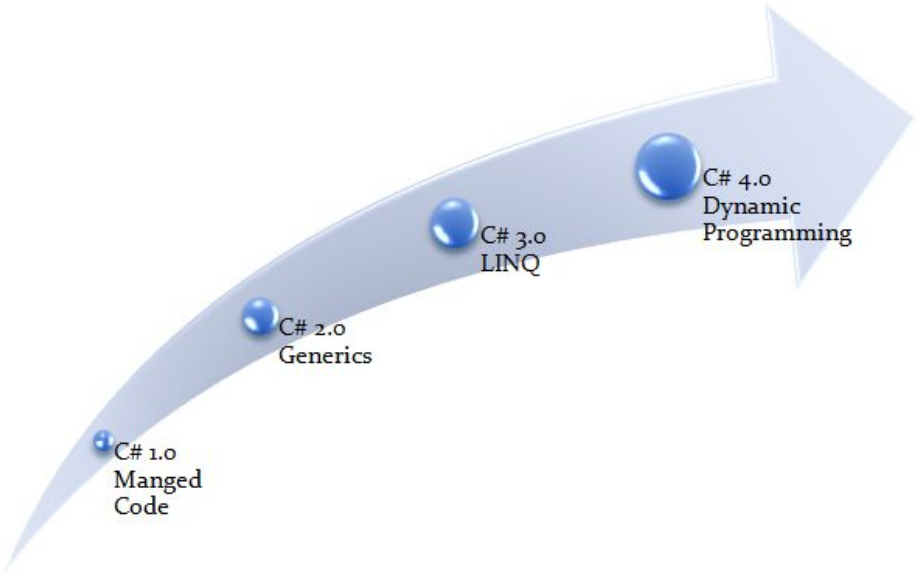
- ***Hejlsberg ’99***
  Start der Entwicklung an **Cool** für .NET
  (**C**-like **O**bject **O**riented **L**anguage)

- ***Juli 2000***
  Professional Developers Conference
  Erste Öffentliche Präsentation des .NET Frameworks, Umbenennung von Cool in **C#**
  (sprich: C Sharp)

Anders Hejlsberg
Technical Fellow und
Chief Architect, C#
Vater von TurboPascal, Delphi

Historie: C# 1.0 – 4.0

- **C# 1.0 (01/2002)**
  wird zusammen mit .NET 1.0 released
- **C# 1.2 (04/2003)**
- **C# 2.0 (11/2005)**
  Generics, Partielle Klassen, Statische Klassen, Anonyme Delegaten, Nullable Types
- **C# 3.0 (11/2007)**
  LinQ, Anonyme Typen, Lambda Ausdrücke, Extension methods, Object/Collection Initializer
- **C# 4.0 (04/2010)**
  Dynamic member lookup, Co-/Contravariance, Named/Optional parameters



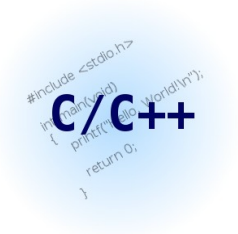
Allgemeine Infos zu C#

- Der Ursprung des Namens C♯:
  Symbol für den in der Musik durch ein Kreuz (♯) um einen Halbton erhöhten Ton C, das **Cis**.
  Englisch: **C sharp**.

- C# Dialekte/Derivate:
  - Spec#
  - Sing#
  - Cω
  - eXtensible C# (XC#)
  - Multiprocessor C# (MC#)
  - Metaphor
  - Polyphonic C#
  - Vala

Allgemeine Infos zu C#

- C# ist standardisiert unter ISO/IEC 23270, ECMA 334
- C# wurde beeinflusst von
  - Java
  - C/C++
  - Pascal/Delphi
  - Haskell
  - Modula-3
  - Visual Basic



- C# kann in folgenden IDEs benutzt werden:
  - Microsoft Visual Studio 2003, 2005, 2008, 2010
  - SharpDevelop
  - MonoDevelop
  - XNA Game Studio
  - C#-Builder
  - Baltie

Inhalt

- 1. Grundlagen
- 2. Sprachdesign und Typisierung
  - Paradigmen in C#
  - Typisierung: Wert- und Verweistypen
  - Ordnungsstrukturen: Namespaces, Dateien, Assemblies
- 3. Sprachfeatures/-syntax

Kleine Umfrage: Wer von euch…

- … hat überhaupt schon einmal programmiert?
- … hat schon einmal in C# programmiert?
- … hat schon kleine Projekte in C# implementiert?
- … hat schon berufliche Erfahrung mit .NET/C# gemacht?
- … hat sich nur in der Tür geirrt?

Paradigmen in C#

C# ist…

Typen in C#

- C# ist eine streng typisierte, objektorientierte Sprache
- Zwei Arten von Typen:
  - Referenz-/Verweistypen (`class`, `interface`, `delegate`, **Arrays,…**)
  - Werttypen (`struct`, `enum`, **Basistypen,…**)
- Jeder Typ erbt automatisch von `object`,
  d.h. alle Typen bieten die Methoden von `object` an (wie `ToString()`)
- Für Referenztypen existiert ein Sondertyp:
  `null` – die leere Referenz

Unterschied: Verweis- und Werttypen

- **Werttypen**
  - Ein Wert-Typ wird auf dem **Stack** gespeichert. Eine Instanziierung ist für eine Instanz eines Wert-Typs nicht notwendig. Eine Zuweisung erzeugt eine **Kopie des Wertes**, bei Vergleichen wird auf **Wertidentität** geprüft.

- **Verweistypen**
  - Ein Verweistyp wird auf dem **Heap** gespeichert, auf dem Stack gibt es einen Verweis auf die Speicheradresse im Heap. Eine Instanziierung ist erforderlich, um eine Instanz nutzen zu können. Eine Zuweisung erzeugt nur eine **Kopie des Zeigers**, bei Vergleichen wird auf die **Identität der Zeiger** geprüft.

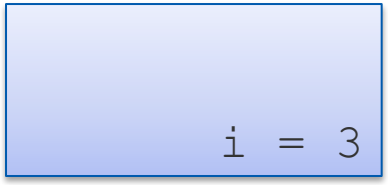
Boxing/Unboxing

Werttypen können mittels Cast in Referenztypen gepackt werden.

```
object o = (object)3;
```

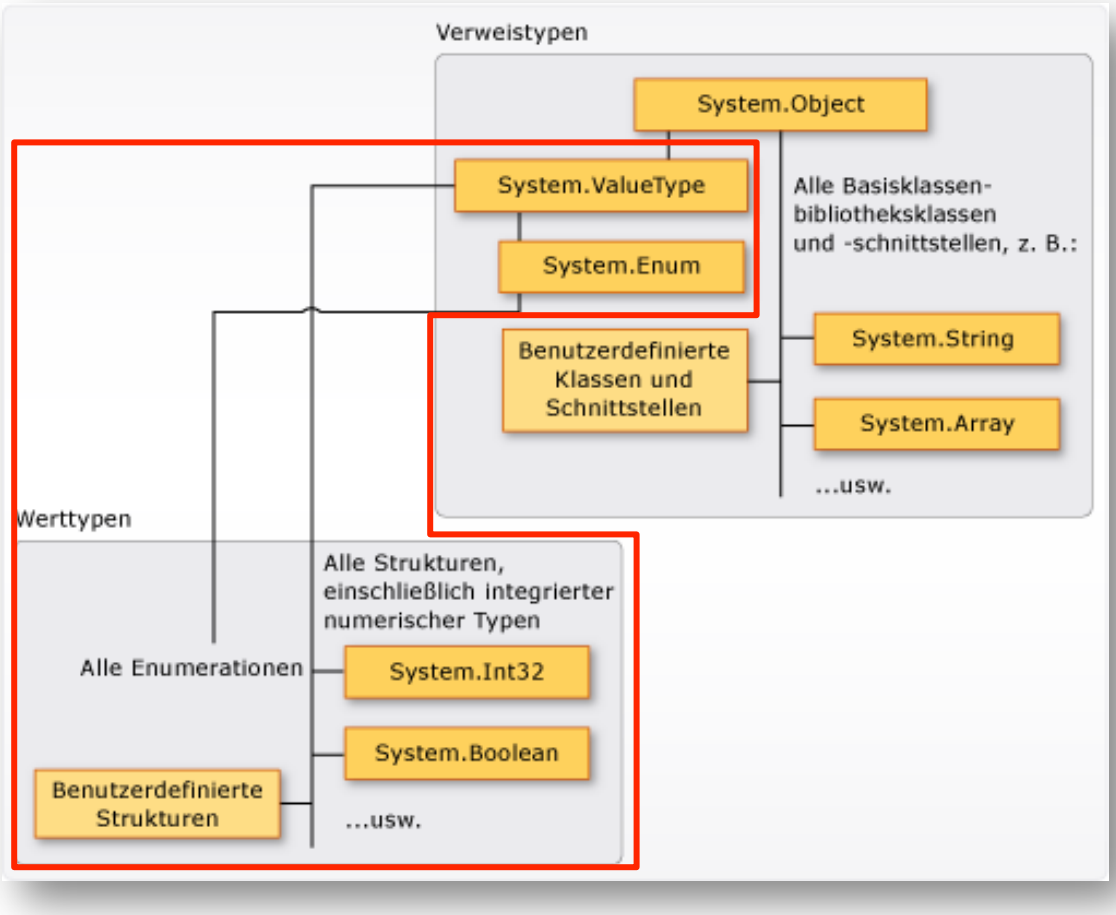
Boxing

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



Unboxing

Typisierung: Werttypen



Werttypen: Basistypen

Datentyp	Wertebereich	Größe in Bits	.NET Laufzeittyp
bool	true oder false	1	System.Boolean
byte	0 … 255	8	System.Byte
sbyte	-128 … 127	8	System.SByte
char	0 … 65535	16	System.Char
short	-2^{15} … $2^{15}-1$	16	System.Int16
ushort	0 … 65535	16	System.UInt16
int	-2^{31} … $2^{31}-1$	32	System.Int32
uint	-32.768 … 32.767	32	System.UInt32
float	$1{,}4 \times 10^{-45}$ … $3{,}4 \times 10^{38}$	32	System.Single
ulong	0 … $2^{64}-1$	64	System.UInt64
long	-2^{63} … $2^{63}-1$	64	System.Int64
double	$5{,}0 \times 10^{-324}$ … $1{,}7 \times 10^{308}$	64	System.Double
decimal	$\pm 1{,}0 \times 10^{-28}$ … $\pm 7{,}9 \times 10^{28}$	128	System.Decimal

Basistypen - Beispiele

```
1  uint u = 6;
2  int i = -10;
3  byte b = 0x01;          // Hexadecimal
4  float f = 4.0F;
5  double d = 0.5D;
6  double d2 = 0.5F;
7
8  char c1 = 'Z';          // Character literal
9  char c2 = '\x0058';     // Hexadecimal
10 char c3 = (char)88;     // Cast from integral type
11 char c4 = '\u0058';     // Unicode
12 char c5 = '\t';         // Special character
13
14 decimal d = 440.5m;
```

Werttypen: Enumerationen

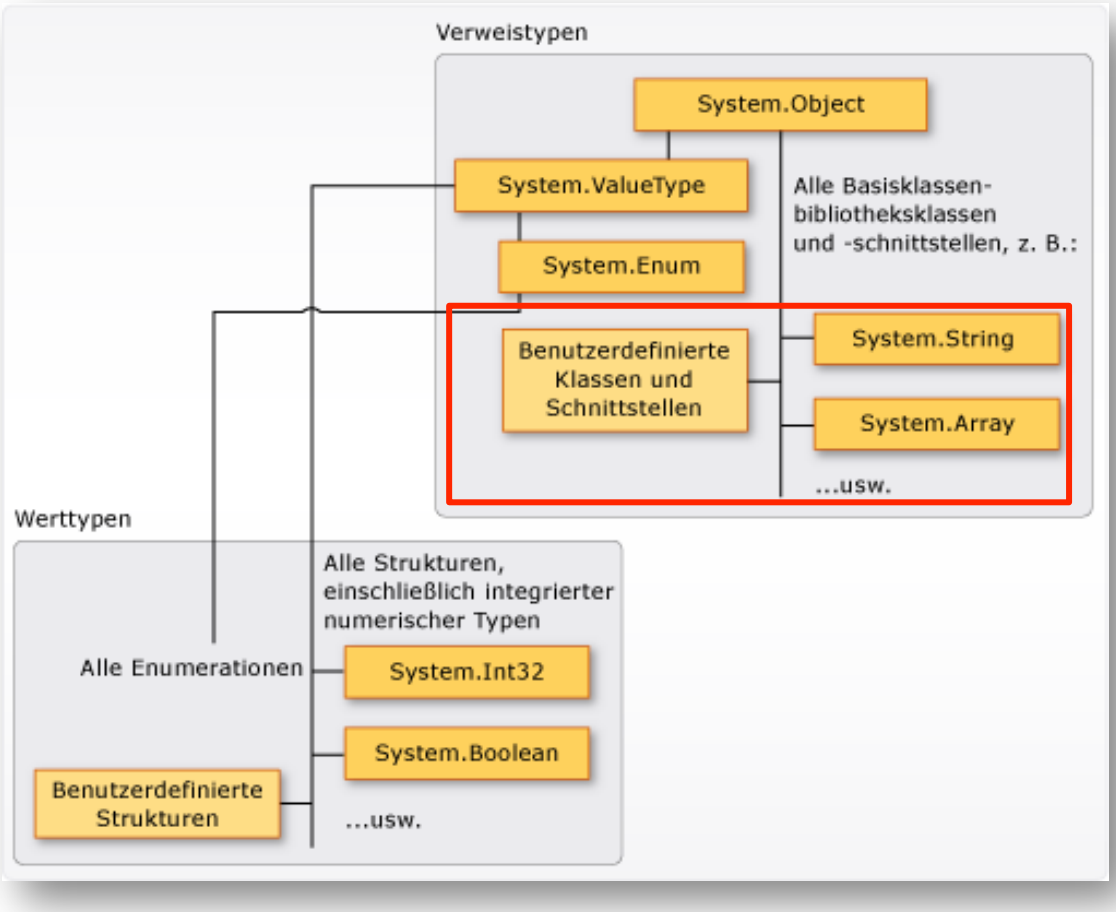
```
1  enum DaysOfWeek
2  {
3      Monday = 1,
4      Tuesday,     // = 2
5      Wednesday,   // = 3
6      Thursday,    // = 4
7      Friday,      // = 5
8      Saturday,    // = 6
9      Sunday       // = 7
10 };
11
12 DaysOfWeek today = DaysOfWeek.Thursday;
13 int today = (int)today; // today = 4
```

Werttypen: Strukturen

```
1  public struct Circle
2  {
3      public double radius;
4      public double centerX;
5      public double centerY;
6  }
```

- Strukturen können auch **Konstruktoren**, **Konstanten**, **Felder**, **Methoden**, **Eigenschaften**, **Indexer**, **Operatoren**, **Ereignisse** und **geschachtelte Typen** enthalten. Wenn allerdings mehrere dieser Member benötigt werden, sollte statt einer Struktur eine **Klasse** verwendet werden.
- Strukturen können eine **Schnittstelle** implementieren, aber sie können nicht von einer anderen Strukturen erben. Aus diesem Grund können Strukturmember nicht als protected deklariert werden.

Typisierung: Verweistypen



Typisierung: Arrays

```
int[] myInts = new int[10];
int[] myInts = new int[10]{0,0,0,0,0,0,0,0,0,0};
```

32 Bit

Wert	0	0	0	0	0	0	0	0	0	0
Index	0	1	2	3	4	5	6	7	8	9

```
myInts[2] = 255;
```

Wert	0	0	255	0	0	0	0	0	0	0
Index	0	1	2	3	4	5	6	7	8	9

```
myInts[7] = myInts[2] - 1;
```

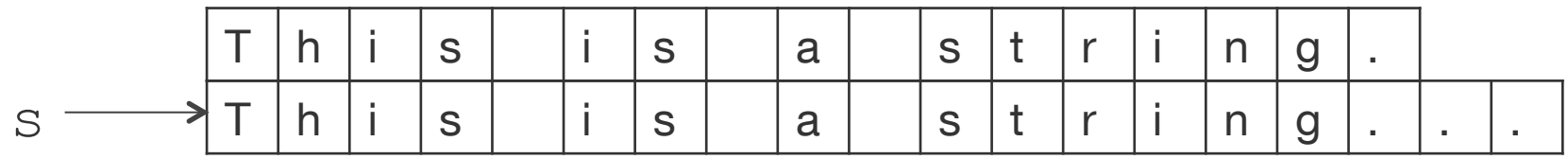
Wert	0	0	255	0	0	0	0	254	0	0
Index	0	1	2	3	4	5	6	7	8	9

Typisierung: Strings

- Strings sind…
  - Besondere Referenztypen
  - Zeichenketten aus Unicode Zeichen, die unveränderlich sind

```
string s = "This is a string."
```


```
s += ".."
```



  - Gleichheit (== und !=) wird bei Strings **nicht** über Referenzgleichheit sondern durch Wertgleichheit überprüft (im Unterschied zu Java, ermöglicht durch Operatorüberladung)

Typisierung: Klassen

```
1  public class Foo : ICloneable, FooBase
2  {
3      private int myInt = 0;
4      public int MyInt
5      {
6          get { return myInt; }
7          set { myInt = value; }
8      }
9
10     internal Foo()
11     {
12
13     }
14 …
15     protected void AddToMyInt(int num)
16     {
17         myInt += num;
18     }
19 }
```

Vererbungsliste

Feld

Property

Konstruktor

Methode

Objekte konstruieren und benutzen

```
1   Foo f = new Foo();                          Konstruktoraufruf
2   f.MyInt = 1;                                Propertyzuweisung
3
4   Foo g = new Foo();                          Konstruktoraufruf
5   g.AddToMyInt(3);                            Methodenaufruf
6
7   f.MyInt = g.MyInt;                          Propertyzuweisung
8
9   g.myInt = 0;                                Feldzuweisung
10
11  System.Console.WriteLine(f.MyInt);          Ausgabe
```

Frage: Welcher Wert wird ausgegeben?

Ordnungsstrukturen: Namespaces

Alle Basisstrukturen werden mit sogenannten Namespaces geordnet.

Namespace **bildet eine hierarchische Ordnungsstruktur, auf die mit dem Operator („.“) zugegriffen werden kann. Namespace ≠ Scope ≠ Assembly!**

Beispiele:

`System.DateTime`

`System.Xml.XmlDocument`

`System.Xml.Serialization.XmlSerializer`

= Namespace

Importieren von Namespaces durch das Schlüsselwort `using`.

Ordnungsstrukturen: Dateien

- **Code wird in *.cs Dateien erfasst.**

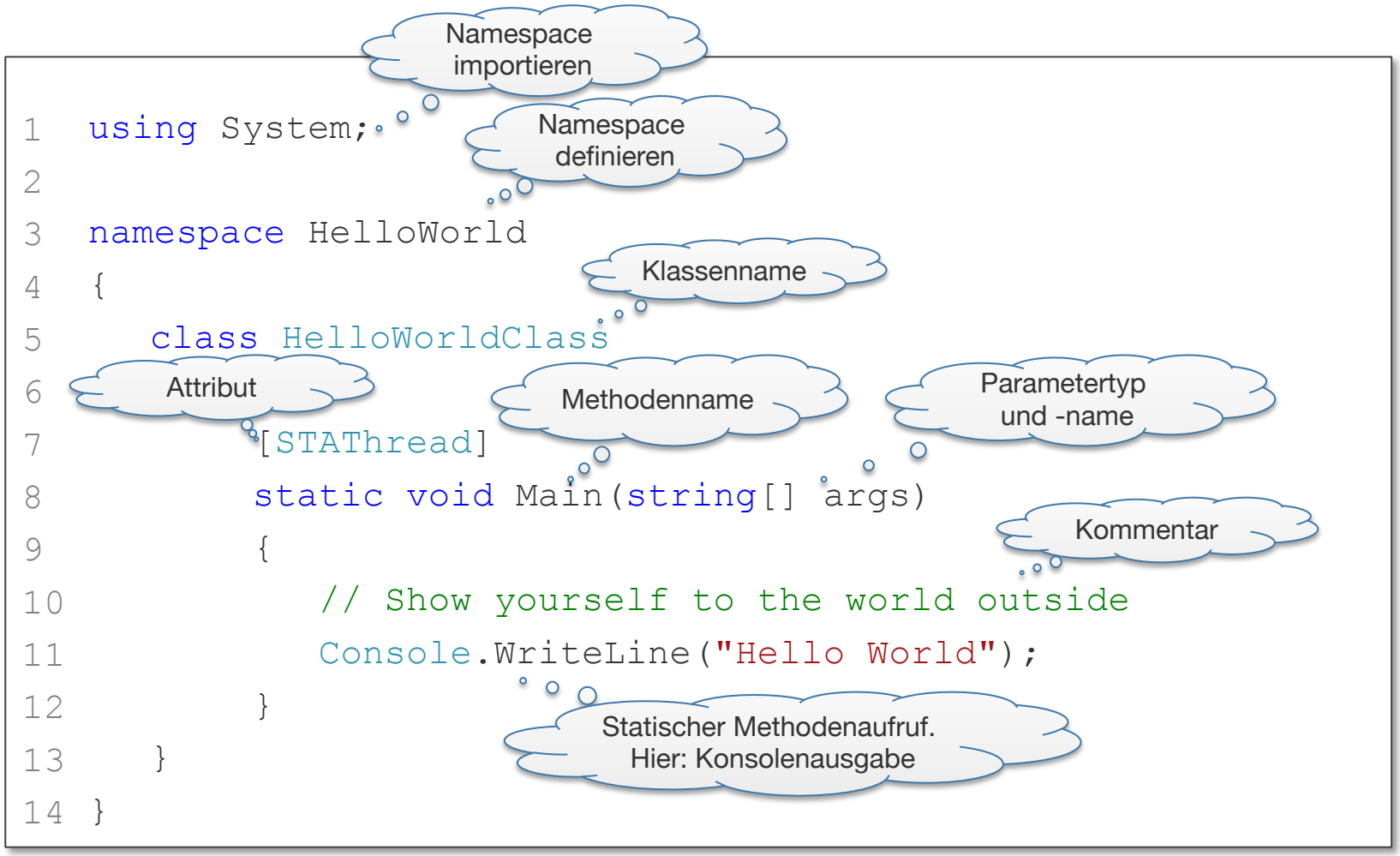
- **Konventionen**
  - **Eine Datei pro Klasse/Struktur**
  - **Klassenname = Dateiname**
  - **Ordnerstruktur = Namespacestruktur**

- **Sonderfall:**
  Klassen können mittels des Schlüsselworts `partial` auf mehrere Dateien verteilt werden.

Inhalt

- 1. Grundlagen
- 2. Sprachdesign
- **3. Sprachfeatures/-syntax**
  - **Hello World!**
  - **Interfaces, Methoden, Delegates, Scopes**
  - **Sichtbarkeit, Schleifen, Generics**
  - **Exceptions, Operatoren**

Hello World in C#




```
1   using System;
2
3   namespace HelloWorld
4   {
5       class HelloWorldClass
6       {
7           [STAThread]
8           static void Main(string[] args)
9           {
10              // Show yourself to the world outside
11              Console.WriteLine("Hello World");
12          }
13      }
14  }
```

Sichtbarkeitsstufen in C#

Stufe	Sichtbarkeit
`public`	Global sichtbar
`internal`	innerhalb der gleichen Assembly
`protected`	innerhalb der erbenden Klasse
`internal protected`	innerhalb der erbenden Klassen in der gleichen Assembly
`private`	nur innerhalb der Klasse

Mehr unter http://de.wikipedia.org/wiki/Sprachelemente_von_C-Sharp

Bedingte Ausführung/Verzweigungen

```
if (<boolescher Ausdruck>)
{
    // something here
}
else if (<boolescher Ausdruck>)
{
    // other things here
}
else
{
    // everything else here
}
```

```
switch (<Variable>)
{
   case Case1:   /* Code here */ break;
   ...
   case CaseN-1: /* Jump to CaseN */
   case CaseN:   /* Code here */ break;
   default:      /* Code here */ break;
}
```

Schleifen

```
while (< boolescher Ausdruck >)
{
    // looped Code here
}
```

```
for (<Init>;<Condition>;<Step>)
{
   // looped code here
}
```

```
do
{
    // looped code here
}
while (<boolescher Ausdruck>);
```

```
foreach (ElementTyp name in Auflistung)
{
   // looped code here
}
```

`break`	gesamte Schleife verlassen
`continue`	Einzelnen Schleifenlauf überspringen

Operatoren in C#

- **Vergleichsoperatoren**

`==`	prüft auf Gleichheit;
`!=`	prüft auf Ungleichheit
`<, >, <=, >=`	prüft auf Erfüllung der entsprechenden Relationen

- **Logische Verknüpfungen**

`!`	*NOT* (unäre Negation)
`&&`	bedingtes *AND*
`||`	bedingtes *OR*

- **Typprüfungen und Casts**

`is`, `typeof`	Typkompatibilität, Typexploration
`as`	Cast (geprüft)
`(<Typ>)`	Cast (ungeprüft, Ausnahme bei Fehler)

Operatoren in C# II

- **Arithmetische Operatoren**

`+, -, *, /`	Grundrechenarten
`++, --`	(Prä- oder Post-) Inkrement, Dekrement
`%`	Modulo

- **Zugriffs-/Zuweisungsoperatoren**

`[<Index>]`	Zugriff auf indexierte Datenstruktur
`=`	Zuweisung
`+=, -=, *=,…`	kombinierte Zuweisung

- **Bitoperatoren**

`<<, >>`	Bitshifting
`&, |, !, ^`	Logisches AND, OR, NOT, XOR

Mehr unter http://de.wikipedia.org/wiki/Sprachelemente_von_C-Sharp

Methoden

```
<Sichtbarkeit> <Rückgabetyp> <Name>(<Parameterliste>)
{
      // Implementierung
}
```

z.B.:

```
1 public string SayHelloWorld()
2 {
3       return "Hello World";
4 }
```

```
1 internal static void CalculateSomething(int x, int y)
2 {
3       x + y * (x - y); // Does not make sense with void
4 }
```

Methoden – Beispiel: Maximum aus zwei Zahlen

```
1  public int Max(int first, int second)
2  {
3      if (first >= second)
4      {
5          return first;
6      }
7      else
8      {
9          return second;
10     }
11 }
```

```
1  public int MaxFunctional(int first, int second)
2  {
3          return first >= second ? first : second;
4  }
```

Delegates

- Können für Referenzen auf Methoden benutzt werden
- Besitzen eine festgelegte Signatur
- Werden u.a. für Ereignisse in C# benutzt

Mehr am 26.01. durch Amin Ahantab.

```
1  delegate string MyStringDelegate(string input);
2
3  public string LowerFunction(string inputStr)
4  {
5          return inputStr.ToLower();
6  }
7
8  public void OtherFunction(string someString)
9  {
10         MyStringDelegate del = LowerFunction;
11         del(someString); // Calls LowerFunction(someString)
12 }
```

Interfaces in C#

```
1  public interface INumberable
2  {
3      int Number { get; set;}
4      string NumberToString();
5  }
6  public class NumerableImplementation : INumberable
7  {
8      private int number = 0;
9      public int Number
10     {
11         get { return number ; }
12         set { number = value; }
13     }
14     public string NumberToString()
15     {
16         return number.ToString();
17     }
18 }
```

Konventionen für Bezeichner in C#

- **CamelCase**
  (z.B. backColor)
  - Felder
  - Parameter
  - Lokale Variablen
- **Pascal Case**
  (z.B. BackColor)
  - Klassen
  - Interfaces
  - Ereignisse
  - Strukturen
  - Properties
  - Enumerationen
  - Enumerationswerte
- **Uppercase**
  (z.B. BACKCOLOR)
  - Für Bezeichner <= 2 Buchstaben, meist in Namespaces: System.IO

Scopes – Gültigkeitsbereiche

```
1  public class Foo
2  {
3      private int myInt = 0;
4      public void DoSomething()
5      {
6          int localVar = 1;
7          if(myInt == 0)
8          {
9              string s = „Hello World“;
10         }
11     }
12 }
```

myInt

localVar

s

Generics

- Generische Klassen:

```
class GenericClass<T> where T : IComparable
{
    T member;
    …
}
```

- Generische Methoden:

```
U GenericMethod<T,U>(T input) where T : new()
{
    T tVal = new T();
    …
}
```

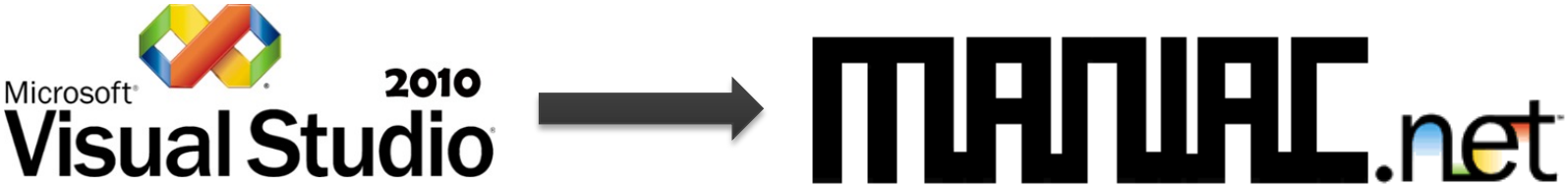
Exceptions und Exceptionhandling

- Exceptions
  - sind Klassen
  - erben direkt oder indirekt von `Exception`
  - Können mittels `throw` geworfen werden
  - Werden in `catch` Blöcken gefangen

Mehr am 17.11. durch Thomas Ladek.

Tipps für den Einstieg in C#

- Am Besten lernt man eine Sprache, indem man sie **benutzt**
- Kostenlose Onlinematerialien:
  - http://openbook.galileocomputing.de/visual_csharp/
  - http://www.guidetocsharp.de/
  - http://msdn.microsoft.com/en-us/library/67ef8sbd(v=VS.100).aspx
  - http://de.wikipedia.org/wiki/Sprachelemente_von_C-Sharp
  - http://www.mycsharp.de/wbb2/thread.php?threadid=78856



https://prod.maniac.tum.de/ManiacGUI

Viel Spaß.