C# – 1

by

Dr. Günter Kolousek

Überblick

- Einflüsse von Java, C++, Delphi (Object-Pascal)
- stark getypt
- statisch und dynamisch getypt
- statische und dynamische Bindung
- single-inheritance
- Sourcecodedateien haben Endung: `.cs`
- *pascal case*

C# Spezialitäten & Sonstiges

- implizit typisierte Variable (Typinferenz)
- Referenzparameter, optionale Parameter, anonyme Typen, nullable Typen, `const`, `readonly`, `sealed`, `override` - `virtual` - `new`, `as`, `default`
- Properties
- Operator overloading
- Indexer (”Überladen des `[]` Operators”)
- Delegates (”getypter Funktionszeiger”)
- Ereignisse (`event`)
- Lambda-Ausdrücke (anonyme Methode)
- Präprozessor
- Attribute (Java: Annotations)

Hello World 1

```
// z.B. in hello.cs
// verwende alles aus namespace System
using System;

public class HelloWorld {
  // auch mit 'String[] args' bzw. 'string[] args'
  // auch int anstatt void; nur 1 Main je Assembly!
  static public int Main() {
      // Console.Out.WriteLine(”Hello, World!”);
      Console.WriteLine(”Hello, World!”);
      return 0;  // exit code
  }
}
```

Hello World 2

```
using System;

public class HelloWorld {
    static public void Main() {
        string name="";  // -> System.String
        while (name == "") {  // beachte: ==
            Console.Write("Your name, please: ");
            // Console.In.ReadLine();
            name = Console.ReadLine();
        }
        object oname=name;  // -> System.Object
        // automatisch ToString(), da:
        //   string operator+(string, object)
        Console.WriteLine("Hello, " + oname + "!");
    }
}
```

eingebaute Aliase, z.B.: `string` (→ Keyword!)

Hello World 3

```
// benutzerdefinierte Aliase
// ArrayList *nicht* verwenden (-> List)!
using AL=System.Collections.ArrayList;
using Con=System.Console;

public class HelloWorld {
  static public void Main() {
    AL al=new AL();
    al.Add("Hello, ");al.Add("World ");al.Add("!");

    foreach (object o in al) Con.Write("{0}", o);

    Con.WriteLine();
    Con.WriteLine(al.Count);  // -> 3
    Con.WriteLine(((string)al[0]).Length); // -> 7
    Con.WriteLine(((string)al[0])[0]);  // -> H
  } }
```

Steueranweisungen

- prinzipiell wie in Java
- `switch`
  - auch mit Strings
  - am Ende jedes `case` (außer `case`-Klausel ist leer!)
    - `break;`
    - `goto case "xxx";` (Annahme: es existiert `case "xxx"`)
    - `goto default;`
- `foreach`

Namespaces

- Wie `package` in Java, aber…
- Kein Zusammenhang von namespace-Hierarchie zu Verzeichnishierarchie
- Eine namespace Deklaration: auch auf mehrere Dateien
- Verschachtelte namespaces möglich

Namespaces – 2

```
// math.cs
namespace math {
    public class Math {
        public static double sqrt(double num) {
            return System.Math.Sqrt(num);
        }
    }

    namespace approx {
        public class Math {
            public static double sqrt(double num) {
                return num / 2;
            }
        }
    }
}
```

Namespaces – 3

```
// calc.cs
using System;
using MathApprox=math.approx.Math;

public class Calc {
    public static void Main() {
        Console.WriteLine(math.Math.sqrt(3));
        // Literale werden geboxt
        Console.WriteLine(MathApprox.sqrt(3) +
                          0.ToString());
    }
}
```

(wichtige) Standard-Namespaces

- `System`
- `System.Collections`
- `System.Collections.Generic`
- `System.Data` → ADO.NET
- `System.IO`
- `System.Linq` → Language Integrated Query
- `System.Net`
- `System.Numerics`
- `System.Reflection`
- `System.Runtime`
- `System.Security`
- `System.Text`
- `System.Threading`
- `System.XML`

Arten von Datentypen

- Wurzel ist `object` (`System.Object`)!
- Werttypen (value types)
  - werden am Stack oder *inline* am Heap abgelegt
    - Heap: wenn in anderem Typ und dieser am Heap!
  - werden als Kopie übergeben
  - Boxing und Unboxing in etwa analog zu Java
  - Achtung: vgl. mit *value object* aus `data_types.pdf`!
- Referenztypen (reference types)
  - werden am Heap gespeichert
  - Referenz wird als Kopie übergeben

Werttypen

- abgeleitet von `System.ValueType`
- 2 Arten: Strukturen und Aufzählungen
- Strukturen
  - Numerische Typen
    - Integrale Typen: `int`, `long`,...
    - Gleitkommatypen: `float`, `double`
    - `decimal`
  - `bool`
  - benutzerdefinierte Strukturen

Überblick über skalare Typen

Typ	Bereich	System
bool	true, false	Boolean
char	[U+0000,U+ffff]	Char
sbyte	$[-128, 127]$	SByte
byte	$[0, 255]$	Byte
short	$[-2^{15}, 2^{15} - 1]$	Int16
ushort	$[0, 2^{16} - 1$	UInt16
int	$[-2^{31}, 2^{31} - 1]$	Int32
uint	$[0, 2^{32} - 1]$	UInt32
long	$[-2^{63}, 2^{63} - 1]$	Int64
ulong	$[0, 2^{64} - 1]$	UInt64

Überblick über skalare Typen – 2

- `float`
  - `System.Single`
  - 32 Bits
  - Bereich: $[-3.4 \cdot 10^{38}, -1.5 \cdot 10^{-45}], [1.5 \cdot 10^{-45}, 3.4 \cdot 10^{38}]$
  - signifikante Stellen: 7
- `double`
  - `System.Double`
  - 64 Bits
  - Bereich: $[-1.7 \cdot 10^{308}, -5 \cdot 10^{-324}], [5 \cdot 10^{-324}, 1.7 \cdot 10^{308}]$
  - signifikante Stellen: 15-16
- `decimal`
  - `System.Decimal`
  - 128 Bits
  - Bereich: $[-7.9 \cdot 10^{28}, 7.9 \cdot 10^{28}]$
  - signifikante Stellen: 28-29
  - Basis 10!

Überblick über skalare Typen – 3

```
using System;
public class Program {
  public static void Main() {
    Console.WriteLine("{0}: bytes={1}, range=[{2},{3}]",
      typeof(float), sizeof(float),
      float.MinValue, float.MaxValue);
    Console.WriteLine("{0}: bytes={1}, range=[{2},{3}]",
      typeof(double), sizeof(double),
      double.MinValue, double.MaxValue);
    Console.WriteLine("{0}: bytes={1}, range=[{2},{3}]",
      typeof(decimal), sizeof(decimal),
      decimal.MinValue, decimal.MaxValue); } }
```

```
System.Single: bytes:4,
  range:[-3,402823E+38,3,402823E+38]
System.Double: bytes:8,
  range:[-1,7976931486232E+308,1,7976931486232E+308]
System.Decimal: bytes:16,
  range:[-7922816251426433759354395033,792281625142643
```

Zahlenliterale

```
long l=123L;  // auch 123l, aber...
long l2=0xcafe;  // keine oktale...
long l3=0x1234_5678_90AB;
uint ui=1234U;
ulong ul=1234UL;
uint bin=0b1101_1110_0011;
float f=0.1f;
double d=0.1;
decimal eur=123.456M;  // -> money
```

Variablen

- lokale Variable *muss* vor Verwendung Wert haben!
- Instanzvariablen werden automatisch initialisert (”Nullwert”)
- Beispiele

```
class Program {
    static int j=1;
    static void Main() {
        int i;  // not initialized
        i=1;  // assignement
        var k=3;  // type: int
        for (int l=1; l<10; l++) {
            int i=123;  // syntax error!
        }
        int j=2;  // not a syntax error!
        string @class=”abc”;  // @ -> escape
        Console.WriteLine(@class);  // -> abc
    } }
```

Variablen – 2

```
using System;
class Program {  // nullable types...
  static void Main() {
    int x1=1;
    // Nullable<int> x2=null;  // for value types!
    int? x2=null;
    Console.WriteLine(x2 + 1 == null);  // -> True
    // be aware of: if-else!
    Console.WriteLine(x1 < x2);  // -> False
    int? x3=x1;  // implicit conversion
    //int x4=x3;  // compiler error
    int x4=(int)x3;  // -> InvalidOperationExc. if (x3==null)
    Console.WriteLine(x4);  // -> 1
    int x5=x3.HasValue ? x3.Value : -1;  // properties
    try {
        Console.WriteLine(x2.Value);
    } catch (System.InvalidOperationException) {
        Console.WriteLine(x2.GetValueOrDefault()); // -> 0
    }
    int x6=x3 ?? -1;  // coalescing operator
  } }
```

Aufzählungen

```
using System;
class Program {
    enum Color : byte {  // default: int
      red, green, blue}
    static void Main() {
        Color c1=Color.red;
        Console.WriteLine(c1);  // -> red
        Console.WriteLine($"{c1:D}");  // -> 0
        int i=(int)c1;
        Console.WriteLine(i);  // -> 0

        Color red;
        if (Enum.TryParse<Color>("red", out red))
            Console.WriteLine(red);  // -> red

        foreach (var c in
          Enum.GetNames(typeof(Color)))
            Console.WriteLine(c);  // -> red...
    } }
```

Aufzählungen – 2

```
using System;
class Program {
    [Flags]
    enum DOW {
        monday=0x01, tuesday=0x02,
        wednesday=0x04, thursday=0x08,
        friday=0x10, saturday=0x20,
        sunday=0x40, weekend=saturday | sunday,
        workday=0x1f,
        allweek=workday | weekend
    }
    static void Main() {
        DOW d=DOW.monday;
        Console.WriteLine(d);  // -> monday
        Console.WriteLine(DOW.weekend);
        // -> weekend
        Console.WriteLine(DOW.monday | DOW.tuesday);
        // -> monday, tuesday
    } }
```

Referenztypen

- alles was keine Werttypen sind
  - eingebaut: `string`, `object`, `dynamic`
  - benutzerdefiniert: Klassen, Interfaces, Delegates

Strings

- Stringliterale
  - `string` für `System.String`
    - `”Ein String mit 3A \x41 \u0041.”`
  - Verbatim String
    - `@”c:\test\x42”` → `c:\test\x42`
    - `\` keine Bedeutung, daher: `@”Verbatim-String mit ””””` → `Verbatim-String mit ”`
  - Format String
    - ```
      string s1=”World”;
      Console.WriteLine($”Hello {s1}”);
      ```
- Vergleich:
  - `”abc”.Equals(”abc”)`, `”abc” == ”abc”`, `”abc”.CompareTo(”abc”) == 0`
    - d.h. `operator==` (und `!=`) ist überschrieben (aber nicht <,...)
  - `object.ReferenceEquals(s1, s2)`
      ```

Strings – 2

- Methoden

```
using static System.Console;
public class Program {
  public static void Main() {
    var msg="a,b,c,d,e";
    WriteLine(msg[0]);  // -> a
    WriteLine(msg.StartsWith("a,b"));  // -> True
    WriteLine(msg.IndexOf(",d"));  // -> 5
    WriteLine(msg.LastIndexOf(","));  // -> 7
    // -> b,c,d,e
    WriteLine(msg.Substring(2, msg.Length - 2));
    WriteLine("  a,b,c,d,e   ".Trim() == msg);  // -> True
    WriteLine("  a,b,c,d,e   ".TrimStart().TrimEnd()
                == msg);  // -> True
    WriteLine(msg.Insert(0, "x,"));  // -> x,a,b,c,d,e
    WriteLine(msg.Replace(",", ";")); // -> a;b;c;d;e
    WriteLine(msg.Remove(2, 2));  // -> a,c,d,e
    // -> [a, b, c, d, e]
    WriteLine("[{0}]", string.Join(", ", msg.Split(",")));
    WriteLine("X{0}X", "abc".PadLeft(5));  // -> X  abcX
  } }
```

Strings – 3

- Statische Methoden
  - `using str = System.String;`
  - `s3 = str.Concat(s1, s2);`
  - `s2 = str.Copy(s1);`
  - `s = str.Join(", ", new str[]{"a","b"}));`
  - `s = str.Format("{0}EUR={1:f}USD",eur,dol);`
    - `{N[,m][:fmt]}`
    - `m` ... Feldbreite, negativ → linksbündig
    - `fmt` ... Formatierungscode
  - Formatierungscodes, z.B.:
    - `f` Festkomma mit 2 Nachkommastellen
    - `n` mit Tausenderpunkten und 2 Nachkommastellen
    - `p` Prozent: aus 0.25 wird 25%
    - `o` ISO 8601 bei Datum&Zeit
    - `s` "sortable" bei Datum&Zeit (wie ISO 8601, aber ohne Zeitzone und Sekundenbruchteile)
- → `System.Text.StringBuilder`

Strings – 4

```
using System;  using static System.Console;
public class Program {
  public static void Main() {
    var res=string.Format("{0,10:C}", 1234.567M);
    WriteLine(res);  // currency
    WriteLine("{0,10:C2}", 1234.567M);
    WriteLine("{0,10:D6}", -1234);  // decimal
    WriteLine("{0,10:E2}", 1234.567M);  // exponential
    WriteLine("{0,10:F2}", 1234.567M);  // fixed-point
    WriteLine("{0,10:N1}", 1234.567M);  // number
    WriteLine("{0,10:X}", 123456);  // hexadecizmal
    DateTime local_date = DateTime.Now;
    WriteLine("{0:s}", local_date);  } }
€ 1.234,57
€ 1.234,57
   -001234
 1,23E+003
   1234,57
   1.234,6
     1E240
2018-07-18T21:31:58
```

StringBuilder

```
using System;
using System.Text;  // -> StringBuilder
using static System.Console;
public class Program {
  public static void Main() {
    StringBuilder sb=new StringBuilder(”Hello”);
    sb.Append(” World!”);
    WriteLine(sb);  // -> Hello World! (->ToString())
    WriteLine(”{0}, {1}”, sb.Capacity, sb.Length);//-> 16, 20
    sb = new StringBuilder(30);
    WriteLine(sb.Capacity);  // -> 30
    sb.Append(”llx ”);
    sb.Append('W');
    sb.Insert(0, ”He”);
    sb.Replace(”x”, ”o”);
    sb.AppendFormat(”{0}”, ”orld!”);
    WriteLine(sb);  // -> Hello World!
```

StringBuilder - 2

```
            sb = new StringBuilder("---", 30);
            sb.AppendLine();
            sb.Remove(0, 4);
            sb.AppendLine("hello World!");
            sb.Append(true);
            WriteLine(sb[0]);  // -> H
            sb[0] = 'H';
            WriteLine(sb);  // -> Hello World!\nTrue
        }
    }
```

object

- `ToString` ... à la Java
- `GetHashCode`
- `Equals`
  - zusätzlich: statische Methode `Equals(object o1, object o2)` ($\rightarrow$ `null`-Werte!)
- `Finalize` ... wird vom GC automatisch aufgerufen
- `GetType` $\rightarrow$ `System.Type`

```
using System;

class Program {
    static void Main() {
        int i=1;
        var type=i.GetType();
        Console.WriteLine(type);  // System.Int32
        Console.WriteLine(type.IsPrimitive);
        // -> True
    } }
```

dynamic

```
using System;

class Program {
    static void Main() {
        dynamic dyn;
        dyn = 100;
        Console.WriteLine(dyn.GetType() + ": " + dyn);
        // -> System.Int32: 100
        dyn = "abc";
        Console.WriteLine(dyn.GetType() + ": " + dyn);
        // -> System.String: abc
        Console.WriteLine(dyn is string);  // -> True
        try {
            dyn.DoWhatIWant();
        } catch (Exception e) {
            Console.WriteLine(e.Message);
            // -> `string' does not contain a definition
            //  for `DoWhatIWant'
        }
    }
}
```

Arrays

- wie in Java, aber…
- spezielle Syntax zum Deklarieren, Initialisieren und Verwenden
- es wird jeweils eine Subklasse von der abstrakten Klasse `System.Array` angelegt
- eindimensional:

```
using System;
class Program {
    static void Main() {
        int[] arr;
        arr = new int[3];
        int[] arr2=new int[]{1,2,3};
        int[] arr3={4,5,6};
        Console.WriteLine(arr[0]); // -> 0
        foreach (var elem in arr2) {
            Console.Write(elem + " ");
        }
        Console.WriteLine("\n" + arr.Length); // -> 3
    } }
```

Arrays – 2

- 2-dimensional:

```
using System;
class Program {
    static void Main() {
        //              rows x columns
        int[,] mat=new int[3,3];
        mat[0,0] = 1;
        mat[0,1] = 2;
        mat[0,2] = 3;
        Console.WriteLine(mat[0,0]);  // -> 1
        Console.WriteLine(mat[1,1]);  // -> 0
        Console.WriteLine(mat.Length);  // -> 9
        int[,] mat2={
            {1, 2, 3},
            {4, 5, 6}
        };
        Console.WriteLine(mat2.Rank);  // -> 2
        Console.WriteLine(mat2.GetLength(0));  // -> 3
        Console.WriteLine(mat2.GetLength(1));  // -> 3
    } }
```

Arrays – 3

- non-rectangular (jagged)

```
int[][] nonrect={
    new int[]{0},
    new int[]{1,2},
    new int[]{3,4,5},
    new int[]{6,7,8,9}};
WriteLine(nonrect[2][1]); // -> 4
```

- Anlegen und kopieren

```
using System;
class Program {
  static void Main() {
    int[] pos{1,2};
    Array arr = Array.CreateInstance(typeof(int),3);
    arr.SetValue(1, 0);  // arr.SetValue(o, x, y)
    arr.SetValue(2, 1);  // arr.SetValue(o, pos)
    arr.SetValue(3, 2);  // arr.SetValue(o, x, y, z)
    Console.WriteLine(arr.GetValue(1));  // -> 2
    Array arr2=(int[])arr.Clone();  // shallow copy!
  } }  // deep copy needed? -> iterate!
```

Arrays – 4

- Klasse `System.Array`: Basisklasse für Arrays!!
  - Instanzmethoden
    - `Clone`, `Equals`, `CopyTo`,...
  - Statische Methoden, z.B.:
    - `BinarySearch`, `Copy`, `IndexOf`, `Reverse`, `Sort`,...
  - Properties:
    - `Length` ... Anzahl aller Elemente
    - `Rank` ... Anzahl der Dimensionen (bei multi-dimensionalen Arrays)

Arrays – 5

```
using System;
class Program {
    static void Main() {
        string[] names={
            ”hugo”,
            ”mini”,
            ”maxi”,
            ”anne”,
            ”anneliese”
        };
        Array.Sort(names);
        foreach (var n in names)
            Console.WriteLine(n);
    } }
// class? -> interface IComparable
//        -> method CompareTo
```

ValueTuple

- ist ein `ValueType` (i.G.z. `System.Collections.Tupel`)
  - kann gut für Rückgabewerte verwendet werden
  - Vergleich zweier Tupel mit == bzw. ! = erst ab C# 7.3!

```
using System;
using static System.Console;
struct Person {
  public string lname;   public string fname;
  public override String ToString()=>$"{lname} {fname}"; }
public class Program {
 public static void Main() {
   Person p; p.lname = "muster"; p.fname = "maxi";

   var t=("maxi", 2, p); WriteLine($"{t.Item1},{t.Item2},{t.Item3}");

   (string s, int i, Person p) t2=("root", 1, p);
   WriteLine($"{t2.s},{t2.i},{t2.p}");

   var t3=(n:"mini", id:3, p:p); WriteLine($"{t3.n},{t3.id},{t3.p}");

   (var n, var id, var p2) = t3; WriteLine($"{n},{id},{p2}");

   (var n2, _, _) = t3; Console.WriteLine(n2); } }
```