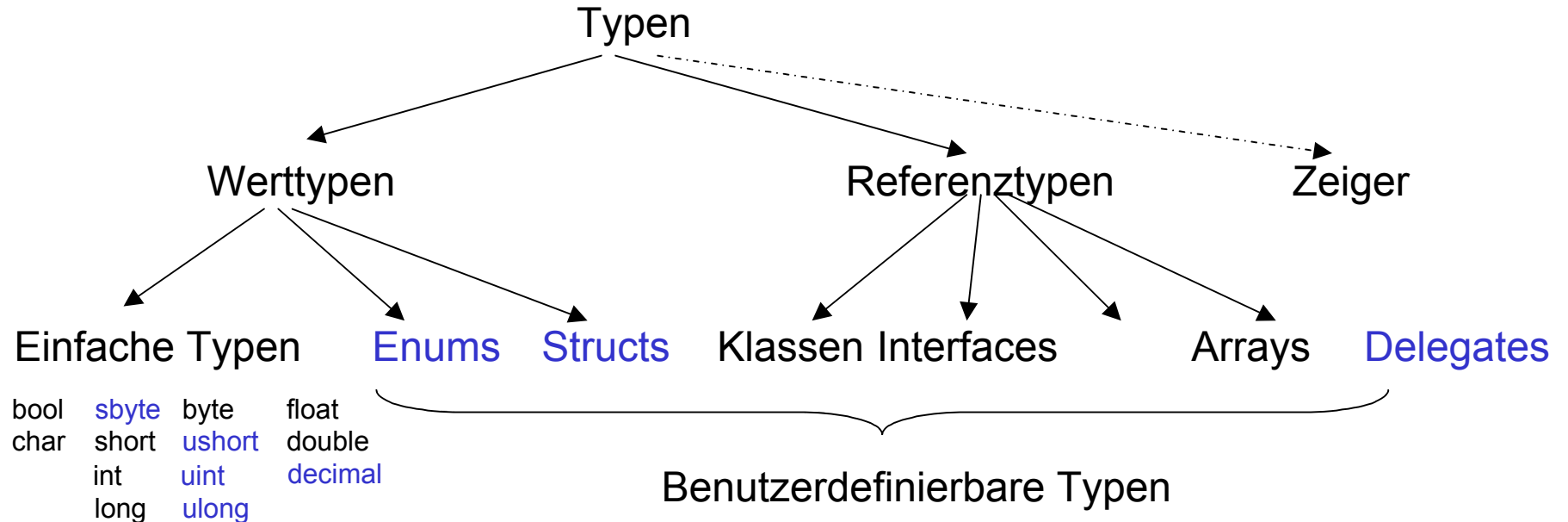



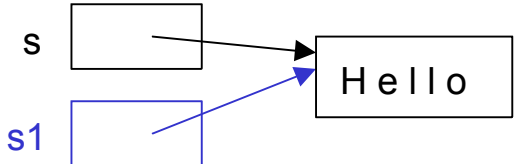
C# Einheitliches Typsystem



Alle Typen sind kompatibel mit object

- Können object-Variablen zugewiesen werden
- Verstehen object-Operationen
- Sind (indirekt) abgeleitet von der Klasse System.ValueType

C# Werttypen versus Referenztypen

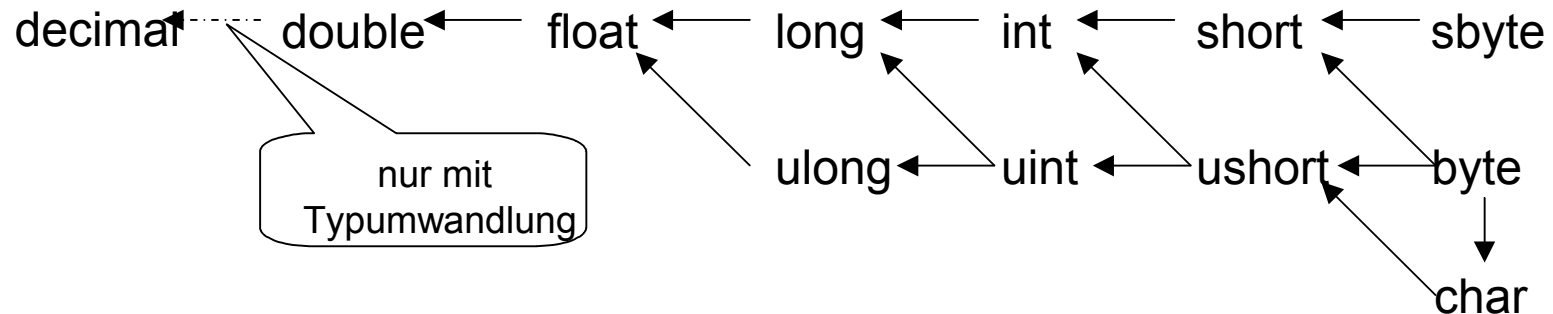
	Werttypen	Referenztypen
Variable enthält	Wert	Zeiger auf ein Objekt
gespeichert am	Stack	Heap
Initialisierung	0, false, '\0'	null
Zuweisung	kopiert Wert	kopiert Zeiger
Beispiel	<pre>int i = 17; int j = i;</pre>  <p>The diagram shows two memory boxes. The top box is labeled 'i' and contains the number '17'. The bottom box is labeled 'j' and also contains the number '17'. Both boxes are outlined in black.</p>	<pre>string s = "Hello"; string s1 = s;</pre>  <p>The diagram shows three memory boxes. On the left, there are two boxes labeled 's' and 's1'. Both 's' and 's1' have arrows pointing to a single box on the right containing the string 'Hello'. The arrow from 's' is black, and the arrow from 's1' is blue.</p>

C# Einfache Typen

Kurzform	Langform	Wertebereich
sbyte	System.SByte	-128 .. 127
byte	System.Byte	0 .. 255
short	System.Int16	-32768 .. 32767
ushort	System.UInt16	0 .. 65535
int	System.Int32	-2147483648 .. 2147483647
uint	System.UInt32	0 .. 4294967295
long	System.Int64	$-2^{63} .. 2^{63}-1$
ulong	System.UInt64	$0 .. 2^{64}-1$
float	System.Single	$\pm 1.5E-45 .. \pm 3.4E38$ (32 Bit)
double	System.Double	$\pm 5E-324 .. \pm 1.7E308$ (64 Bit)
decimal	System.Decimal	$\pm 1E-28 .. \pm 7.9E28$ (128 Bit)
bool	System.Boolean	true, false
char	System.Char	Unicode-Zeichen

C# Implizite und Explizite Datenkonvertierung

Implizit:



Die implizite (ohne spezielle Methoden) Datentypkonvertierung ist nur dann Möglich, wenn dabei keine Informationen verloren gehen können

Beispiel:

```

byte b = 10;
int i = 1000;
i = b; // → implizite Konvertierung (automatisch)
  
```

Explizit:

```

b = (byte) i; // → explizite Konvertierung (Casting), d.h. der Zieltyp muss in
// Klammern davor geschrieben werden
  
```

Übung: GetType() und FullName()

C# Boxing und Unboxing

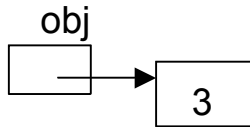
Manchmal ist es notwendig, einen Werttyp in einen Referenztyp object umzuwandeln bzw. umgekehrt einen Referenztyp in einen Werttyp zu konvertieren

Boxing

Bei der Zuweisung

```
object obj = 3;
```

wird der Wert 3 in ein Heap-Objekt eingepackt



```
class Templnt {  
    int val;  
    Templnt(int x) {val = x;}  
}
```

```
obj = new Templnt(3);
```

Unboxing

Bei der Zuweisung

```
int x = (int) obj;
```

wird der eingepackte int-Wert wieder ausgepackt

C# Boxing und Unboxing

In der Regel ist diese Konvertierung notwendig um einen Wertparameter an eine Methode zu übergeben, die eigentlich einen Referenztyp erwartet, oder bei der Verarbeitung des Rückgabewertes einer Methode.

Der Vorgang der Konvertierung eines Wert- in einen Referenztyps wird Boxing genannt.

Der umgekehrte Vorgang der Konvertierung eines Referenz- in einen Werttyps heißt Unboxing und erfordert die Angabe eines Cast.

Das Boxing wird implizit (automatisch) durchgeführt.

Beispiel:

```
int i = 10;  
object IntObj;  
Intobj = i;  
i = (int) IntObj;
```

C# Boxing und Unboxing

Beispiel:

```
class Queue {  
    ...  
    public void Enqueue(object x) {...}  
    public object Dequeue() {...}  
    ...  
}
```

Klasse Queue kann für Referenz- und Werttypen verwendet werden:

```
Queue q = new Queue();  
  
q.Enqueue(new Rectangle());  
q.Enqueue(3);  
  
Rectangle r = (Rectangle) q.Dequeue();  
int x = (int) q.Dequeue();
```

Übung: Addition zweier (beliebiger) Objekte in einer Methode

C# Enumerations

Aufzählungstypen aus definierten Konstanten

Deklaration (auf Namespace-Ebene)

```
enum Color {red, blue, green} // Werte: 0, 1, 2
enum Access {personal=1, group=2, all=4}
enum Access1 : byte {personal=1, group=2, all=4}
```

Verwendung

```
Color x = Color.blue; // enum-Konstanten müssen qualifiziert werden
enum Aktion {Loeschen, Einfuegen, Aendern};
void Ausgabe(Aktion modus)
{
    switch (modus)
    {
        case Aktion.Loeschen: <tue dies> ; break
        case Aktion.Einfuegen: <tue das>; break
    }
}
```

C# Operationen mit Enumerationen

Erlaubte Operationen

Vergleiche	if (c == Color.red) ... if (c > Color.red && c <= Color.green) ...
+, -	c = c + 2;
++, --	c++;

Es wird dabei nicht geprüft, ob der erlaubte Wertebereich über-/unterschritten wird.

- Enumerationen sind nicht zuweisbar an int (außer mit TypeCast)
- Enumstypen erben alle Eigenschaften von object (Equals, ToString, ...)
- Klasse **System.Enum** stellt zusätzliche Operationen auf Enumerationen bereit (GetName, Format, GetValues, ...)

Übung: Farbschema

C# Arrays

Eindimensionale Arrays

```
int[] a = new int[3]; //Klammerpaar muss immer direkt hinter dem Datentyp stehen
int[] b = new int[] {3, 4, 5};
int[] c = {3, 4, 5};
Class1[] d = new Class1[10]; // Array von Referenzen
Class2[] e = new Class2[anzahl]; // Arraygröße kann auch über eine Variable fest-
// gelegt werden (→ dynamisch großes Array)
```

Mehrdimensionale Arrays ("ausgefranst", jagged)

```
int[][] a = new int[2][]; // unterschiedliche Anzahl von Elementen pro Dimension
a[0] = new int[] {1, 2, 3}; // Dimensionen können nur einzeln initialisiert werden
a[1] = new int[] {4, 5, 6, 7};
```

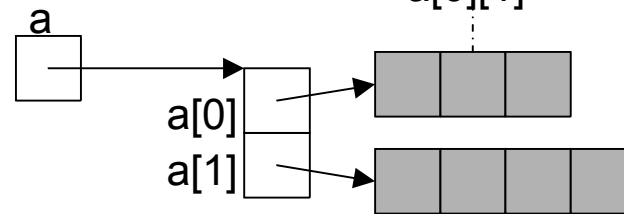
Mehrdimensionale Blockarrays (rechteckig)

```
int[,] a = new int[2, 3]; // Block-Matrix
int[,] b = {{1, 2, 3}, {4, 5, 6}}; // Können direkt initialisiert werden
int[,] c = new int[2, 4, 2];
```

C# Arrays

Ausgefranst (unregelmäßige Arrays, jagged Arrays) $a[0][1]$

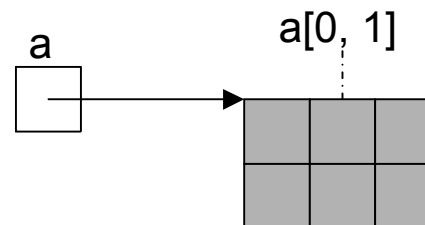
```
int[][] a = new int[2][];
a[0] = new int[3];
a[1] = new int[4];
int x = a[0][1];
```



Jede Dimension muss dabei separat erzeugt werden!

Rechteckig (kompakter, effizienterer Zugriff)

```
int[,] a = new int[2, 3];
int x = a[0, 1];
```



C# Arrays

- Arrays sind abgeleitet von der Klasse System.Array
- Indizierung beginnt bei 0
- Größe des Arrays kann auch über den Wert einer Variablen bestimmt werden
- die Größe des Arrays lässt sich im Nachhinein nicht ändern

Eigenschaften und Methoden der Klasse Array

- **Sort()** sortiert die Elemente eines Arrays
`Array.Sort(b);`
- **Reverse()** kehrt die Reihenfolge der Elemente im Array um
- **Copy()** kopiert einen Teil eines Arrays in ein anderes Array
`int[] a = {7, 2, 5};`
`int[] b = new int[2];`
`Array.Copy(a, b, 2); // kopiert a[0..1] nach b`
- **Clear()** setzt den angegebenen Bereich auf 0 bzw. null
- **Resize()** erstellt ein neues Array und überträgt die Elemente des Ausgangsarray
- **BinarySearch()** durchsucht eines sortiertes Array nach einem Objekt

...

C# Arrays

Eigenschaften und Methoden der Klasse Array

- **Length**: liefert die Länge eines Arrays (aller Dimensionen)

```
int[] a = new int[3];  
Console.WriteLine(a.Length); // → Ausgabe 3  
int[,] c = new int[3, 4];  
Console.WriteLine(c.Length); // → Ausgabe 12  
int[][] b = new int[3][];  
b[0] = new int[4];  
Console.WriteLine("{0}, {1}", b.Length, b[0].Length); // → Ausgabe 3, 4
```
- **GetLength()**: liefert die Größe der als Parameter übergebenen Dimension

```
Console.WriteLine("{0}, {1}", c.GetLength(0), c.GetLength(1)); // → Ausgabe 3, 4
```
- **Rank**: liefert die Anzahl der Dimensionen zurück

```
Console.WriteLine("Dimensionen: {0}", c.Rank); // → Ausgabe 2
```

...

C# ArrayList

Variabel lange Arrays

wichtige Methoden der Klasse ArrayList:

<i>Add()</i>	hängt ein übergebenes Objekt am Ende der Liste an
<i>AddRange()</i>	hängt eine Kollektion von Objekten am Ende der Liste an
<i>BinarySearch()</i>	durchsucht die Liste nach dem Objekt und liefert Index zurück
<i>Clear()</i>	entfernt alle Elemente aus der Liste
<i>Contains()</i>	liefert wahr zurück, wenn sich das übergebene Objekt in der Liste befindet
<i>Insert()</i>	fügt ein übergebenes Objekt an der übergebenen Position ein
<i>InsertRange()</i>	fügt eine übergebene Kollektion von Objekten an der übergebenen Position ein
<i>Remove()</i>	entfernt ein übergebenes Element aus der Liste
<i>RemoveAt()</i>	entfernt ein übergebenes Element an der übergebenen Position aus der Liste
<i>RemoveRange()</i>	entfernt eine übergebene Kollektion von Objekten ab der übergebenen Position aus der Liste
<i>Reverse()</i>	dreht die Reihenfolge der Elemente um
<i>Sort()</i>	sortiert die Elemente in der Liste
<i>Count()</i>	gibt die Anzahl der Elemente zurück
<i>TrimToSize()</i>	vermindert die Kapazität der Liste und passt sie an die Anzahl der Objekte an

C# ArrayList

Beispiel mit einem variabel langen Array

```
using System;
using System.Collections;

class Test {

    static void Main() {
        ArrayList a = new ArrayList();
        a.Add ("Caesar");
        a.Add ("Dora");
        a.Add ("Anton");
        a.Sort ( );
        for (int i = 0; i <a.Count ; i++)
            Console.WriteLine(a[i]);
    }
}
```

Ausgabe :

Anton
Caesar
Dora

C# Klasse System.String

Benutzbar als Standardtyp **string** (=Alias für die Klasse)

```
string s = "Alfonso";
```

Anmerkungen

- Strings sind nicht modifizierbar, d.h. sie enthalten **konstante** Zeichenketten (dazu *StringBuilder*).
Bei allen Operationen an einem string wird stets eine Kopie des Objekts zurückgegeben!
- Können mit + verkettet werden: **"Don " + s**
- Können indiziert werden: **s[i]**
- Längenprüfung: **s.Length**
- Referenztyp, daher Zeigersemantik in Zuweisungen
- aber Wertevergleich mit == und != **if (x == "Alfonso") ...**
- Klasse String definiert viele nützliche Operationen:
CompareTo, CompareOrdinal, IndexOf, StartsWith, Substring, ...

C# Klasse System.String - Initialisierung

durch eine Wertzuweisung:

```
string s1 = "Hello World";  
string s2 = ""; //Leerstring
```

über den Konstruktor:

```
string s3 = new String(' ',20); //mit 20 Leerzeichen
```

mit Hilfe der Empty-Eigenschaft:

```
string s4 = String.Empty;
```

C# Klasse System.String - Länge

durch eine Eigenschaft Length

```
Console.WriteLine("Der String " + s1 + " ist " + s1.Length + " Zeichen lang");
```

über die Methode IsNullOrEmpty():

```
if (String.IsNullOrEmpty(s2))  
    Console.WriteLine("Der String " + s2 + " ist leer oder null");
```

über die Eigenschaft Empty:

```
if (s2==String.Empty)  
    Console.WriteLine("Der String " + s2 + " ist leer");
```

C# Klasse System.String – Zugriff auf die Elemente

auf die Elemente eines Strings kann wie bei einem Array zugegriffen werden, indem ein Index zwischen 0 und Stringlänge-1 angegeben wird.

Ein Zeichen kann dabei aber nicht verändert werden!

Mit Hilfe einer for-Schleife lässt sich der gesamte String zeichenweise durchlaufen.

Zur Ausgabe der einzelnen Zeichen muss die Methode ToString() verwendet werden (ansonsten wird der Zahlencode des Unicode-Zeichens ausgegeben bzw. aufaddiert!)

```
Console.WriteLine ( s1[1].ToString() + s1[2].ToString() + s1[3].ToString() );
```

C# Klasse System.String – Verkettung

mit +

damit lassen sich nicht nur Strings sondern auch beliebige Objekte miteinander verknüpfen, allerdings muss mindestens ein String darunter sein. Die Objekte werden dabei implizit in einen String konvertiert.

```
int z1 = 9, z2 = 17;  
string s1 = "Zahl: " + z1;           //→ ergibt Zahl: 9  
string s2 = z1 + z2;                 //→ ergibt Fehler  
string s3 = z1 + " " + z2;          //→ 9 17  
Console.WriteLine( z1 + z2);        //→ gibt 26 aus
```

C# Klasse System.String – Verkettung

mit **Concat**

der Methode Concat() kann man beliebig viele Objekte übergeben:

```
string s1 = "Montag", s2="";  
int tag = 4, jahr = 2006;  
s2 = String.Concat(s1," ", tag.ToString(),". September ",jahr.ToString());  
//→ ergibt "Montag, 4.September 2006"
```

C# Klasse System.String – Aufteilung

mit **Split**

mit Hilfe der Methode Split kann ein Strings in mehrere Teilstrings zerlegt werden. Dazu übergibt man ihr ein char-Feld mit den möglichen Trennzeichen:

```
string s1 = "Anna,Zirngibl,Baderstr 10,Tattenhausen";  
char t[] = { ',' , ';' };  
string [] s2 = s1.Split(t);  
foreach ( string temp in s2 )  
    Console.WriteLine(temp);
```

C# Klasse System.String – Kopieren

mit **Copy**

die Funktion liefert eine Kopie des Strings zurück:

```
string s1 = "Hello World";  
string s2 = String.Copy(s1);
```

die Funktion Clone erstellt dagegen keine Kopie sondern liefert *nur einen weiteren Verweis* auf den String zurück:

```
string s3 = s1.Clone().ToString();
```

C# Klasse System.String – Teilstring

mit **Substring**

die Methode Substring gibt einen Teilstring einer Zeichenkette zurück. Der Startindex und optional die Länge wird übergeben:

```
string s1 = "Hello World";  
string s2 = s1.Substring(0,5); // → s2 enthält "Hello"
```

C# Klasse System.String – Suchen

mit **IndexOf**

die Funktion gibt den 0-basierenden Index des gefundenen Zeichens bzw. Teilstrings zurück. Optional kann noch ein Startindex ab dem gesucht werden soll, sowie eine Länge übergeben werden:

```
string s1 = "Hello World";  
int pos1 = s1.IndexOf('o'); // → ergibt 4  
int pos2 = s1.IndexOf("or"); // → ergibt 7
```

der Funktion **IndexOfAny** wird ein char-Array übergeben, das mehrere Zeichen enthält, nach denen gesucht werden soll:

```
int pos3 = s1.IndexOfAny(new Char[] {'e','o'}); // → ergibt 1
```

die Funktion **LastIndexOf** findet das letzte Vorkommen eines Zeichens (Suche beginnt am Ende des Strings)

die Funktionen **StartsWith** und **EndsWith** prüfen, ob ein String mit einer bestimmten Zeichenfolge beginnt oder endet

```
if (s1.StartsWith("He")) ... // → liefert wahr
```

C# Klasse System.String – Vergleich

mit den Vergleichsoperatoren `==` und `!=`

dabei wird zwischen Klein- und Großschreibung unterschieden :

```
string s1 = "Montag";  
string s2 = "MONTAG";  
if (s1==s2) ... // → ist falsch
```

mit der Funktion **CompareTo**

```
string s1 = "Maier";  
string s2 = "Meier";  
if (s1.CompareTo(s2)<0) .... // → ist wahr, Funktion liefert -1  
if (s2.CompareTo(s1)<0) .... // → ist falsch, Funktion liefert +1
```

Anmerkung:

um einen caseinsensitiven Vergleich durchführen zu können, müssen die Strings zuvor in Klein- oder Großbuchstaben umgewandelt werden!

C# Klasse System.String – Trimmen und Ausrichten

mit der Funktion **Trim**

die Funktion entfernt führende und abschließende Whitespace-Zeichen bzw die optional übergebenen Zeichen:

```
string s1 = " Montag ";  
s1.Trim(); // → ergibt "Montag"  
string s2 = ",Montag,";  
s1.Trim(new char [] { ',' , ';' }); // → ergibt "Montag"
```

ebenso möglich: **TrimStart** und **TrimEnd**

Die Funktionen **PadRight** und **PadLeft** geben für den String eine feste Länge vor und definieren optional führende bzw. abschließende Füllzeichen

```
string s1 = "Stundenplan";  
char z = '.';  
Console.WriteLine(s1.PadRight(20,z)); // → liefert "Stundenplan....."  
Console.WriteLine(s1.PadLeft(20,z)); // → liefert ".....Stundenplan"
```

C# Klasse System.String – Löschen und Einfügen

Löschen - mit der Eigenschaft **Empty**

```
s1 = String.Empty;
```

- mit der Funktion **Remove**:

```
string s1 = " Stundenplan! ";  
s1.Remove(7,4); // → liefert "Stunden!"
```

zu beachten: Die Anweisung `s1 = null` löscht zwar auch den String, gibt ihn aber auch zusätzlich frei!

Einfügen - mit der Funktion **Insert**

```
string s1 = " Herr Maier ";  
s1.Insert(5,"Dr. "); // → liefert "Herr Dr. Maier"
```

C# Klasse System.String – Ersetzen und Konvertieren

Ersetzen - mit der Funktion **Replace**

```
string s1 = " Herr Dr. Maier ";  
Console.WriteLine (s1.Replace("Dr","Dipl-Ing")); // → liefert "Herr Dipl-Ing. Maier"
```

Konvertieren mit **ToUpper** und **ToLower**

```
string s1 = " Herr Dr. Maier ";  
Console.WriteLine (s1.ToUpper()); // → liefert "HERR DR. MAIER"
```

C# Klasse System.String – Formatierung

Formatierungszeichen und ihre Bedeutung (1)

C,c	Währung (engl. <i>Currency</i>), formatiert den angegebenen Wert als Preis unter Verwendung der landesspezifischen Einstellungen.
D,d	Dezimalzahl (engl. <i>Decimal</i>), formatiert einen Gleitkommawert. Die Präzisionszahl gibt die Anzahl der Nachkommastellen an.
E,e	Exponential (engl. <i>Exponential</i>), wissenschaftliche Notation. Die Präzisionszahl gibt die Nummer der Dezimalstellen an. Der Buchstabe „E“ im ausgegebenen Wert steht für „mal 10 hoch“.
F,f	Gleitkommazahl (engl. <i>fixed Point</i>), formatiert den angegebenen Wert als Zahl mit der durch die Präzisionsangabe festgelegten Anzahl an Nachkommastellen.
G,g	Kompaktformat (engl. <i>General</i>), formatiert den angegebenen Wert entweder als Gleitkommazahl oder in wissenschaftlicher Notation. Ausschlaggebend ist, welches der Formate die kompaktere Darstellung ermöglicht.
N,n	Numerisch (engl. <i>Number</i>), formatiert die angegebene Zahl als Gleitkommazahl mit Kommas als Tausender-Trennzeichen. Das Dezimalzeichen ist der Punkt.

C# Klasse System.String – Formatierung

Formatierungszeichen und ihre Bedeutung (2)

X,x	Hexadezimal, formatiert den angegebenen Wert als hexadezimale Zahl. Der Präzisionswert gibt die Anzahl der Stellen an. Eine angegebene Zahl im Dezimalformat wird automatisch ins Hexadezimalformat umgewandelt.
#	Platzhalter für eine führende oder nachfolgende Leerstelle
0	Platzhalter für eine führende oder nachfolgende 0
.	Der Punkt gibt die Position des Dezimalpunkts an.
,	Jedes Komma gibt die Position eines Tausendertrenners an.
%	Ermöglicht die Ausgabe als Prozentzahl, wobei die angegebene Zahl mit 100 multipliziert wird

C# Klasse System.String – Formatierung

Formatierungszeichen und ihre Bedeutung (2)

X,x	Hexadezimal, formatiert den angegebenen Wert als hexadezimale Zahl. Der Präzisionswert gibt die Anzahl der Stellen an. Eine angegebene Zahl im Dezimalformat wird automatisch ins Hexadezimalformat umgewandelt.
#	Platzhalter für eine führende oder nachfolgende Leerstelle
0	Platzhalter für eine führende oder nachfolgende 0
.	Der Punkt gibt die Position des Dezimalpunkts an.
,	Jedes Komma gibt die Position eines Tausendertrenners an.
%	Ermöglicht die Ausgabe als Prozentzahl, wobei die angegebene Zahl mit 100 multipliziert wird

C# Klasse System.Text.StringBuilder

```

public sealed class StringBuilder {           // namespace System.Text
    publicStringBuilder();                   // Anfangskapazität = 16, wächst dynamisch
    publicStringBuilder (int initCapacity);
    publicStringBuilder (string s);
    publicStringBuilder (string s, int from, int len);

    public int Length
    public int Capacity
    public char this[int i]

    public StringBuilder Append(T x);        // T ... string oder numerischer Typ
    public StringBuilder Insert (int pos, T x);
    public StringBuilderRemove (int pos, int len);
    public StringBuilder Replace(T x, T y);  // T ... string, char
    public bool Equals (object x);
    public string ToString();
    ...
}

```

Beispiel

```

StringBuilder b = new StringBuilder("A.C");
b.Insert(2, "B.");
b.Replace('.', '/');
Console.WriteLine(b.ToString()); // A/B/C

```